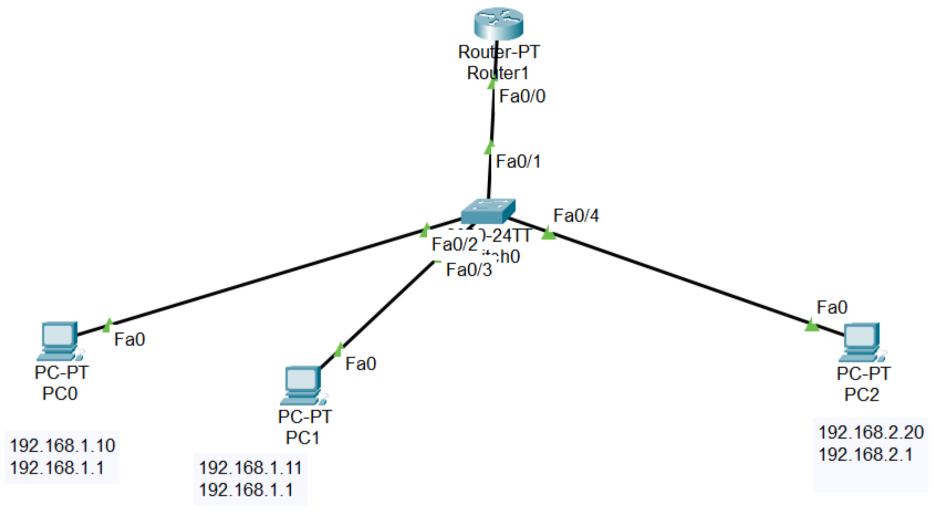
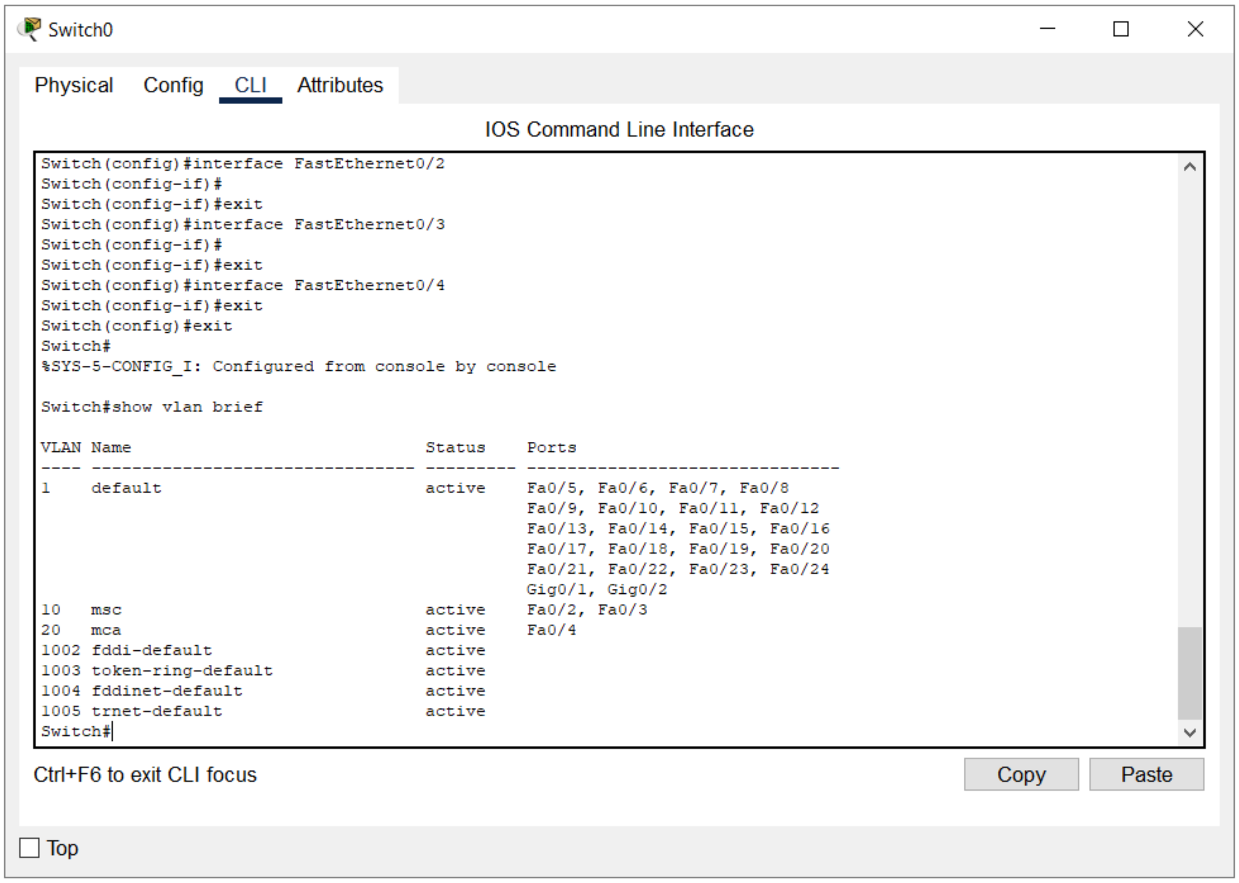
Index

Aim : Implement Inter-VLAN Routing.

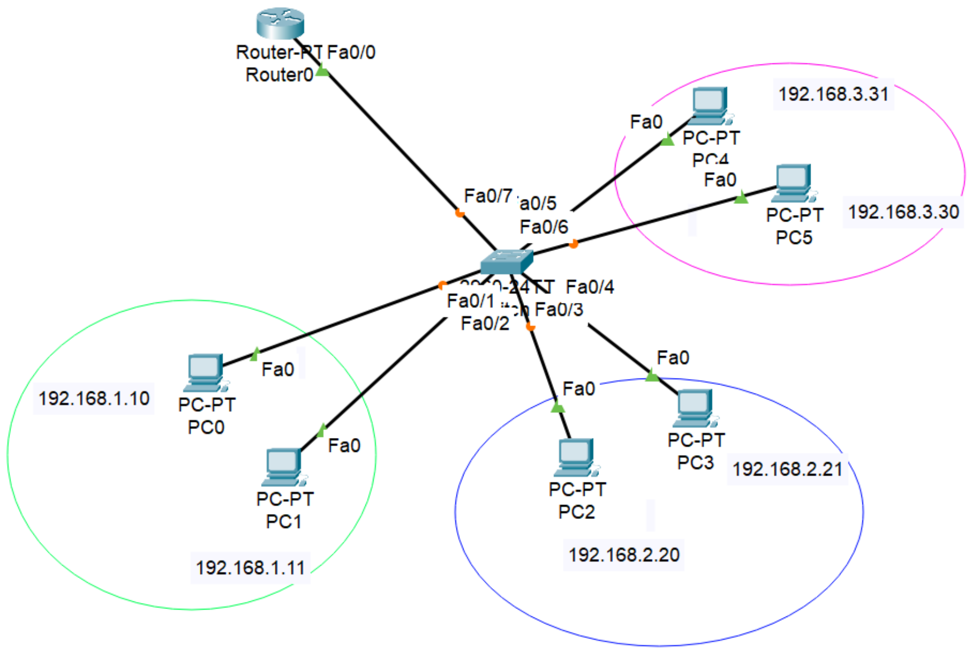
Two vlans

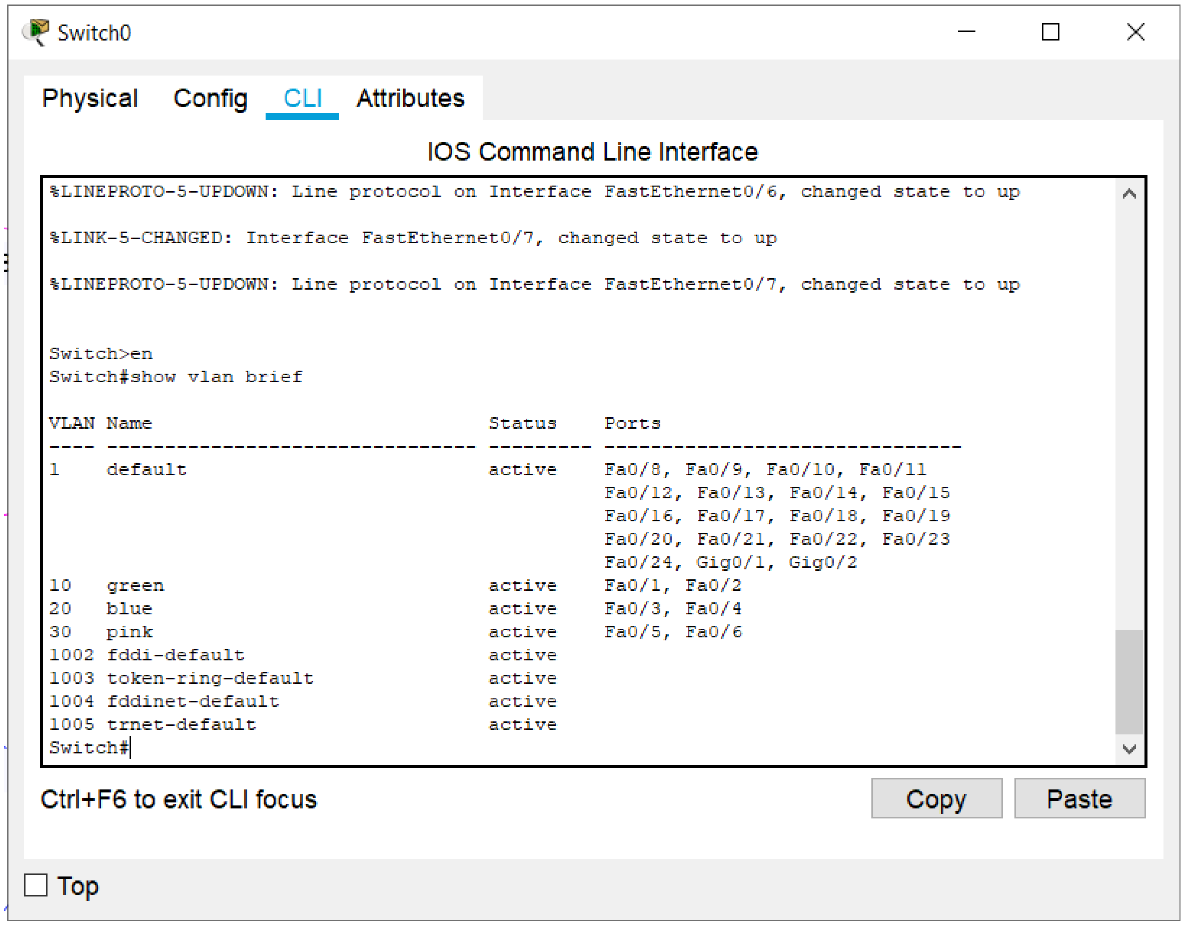




```
Switch(config)#interface FastEthernet0/2
Switch(config-if)#
Switch(config-if)#exit
Switch(config)#interface FastEthernet0/3
Switch(config-if)#
Switch(config-if)#exit
Switch(config)#interface FastEthernet0/4
Switch(config-if)#exit
Switch(config)#exit
Switch#
%SYS-5-CONFIG_I: Configured from console by console

Switch#show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                                Fa0/13, Fa0/14, Fa0/15, Fa0/16
                                                Fa0/17, Fa0/18, Fa0/19, Fa0/20
                                                Fa0/21, Fa0/22, Fa0/23, Fa0/24
                                                Gig0/1, Gig0/2
10   msc                              active    Fa0/2, Fa0/3
20   mca                              active    Fa0/4
1002 fddi-default                     active
1003 token-ring-default               active
1004 fddinet-default                  active
1005 trnet-default                    active
Switch#
```

Three vlans





```
%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/6, changed state to up

%LINK-5-CHANGED: Interface FastEthernet0/7, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/7, changed state to up


Switch>en
Switch#show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/8, Fa0/9, Fa0/10, Fa0/11
                                                Fa0/12, Fa0/13, Fa0/14, Fa0/15
                                                Fa0/16, Fa0/17, Fa0/18, Fa0/19
                                                Fa0/20, Fa0/21, Fa0/22, Fa0/23
                                                Fa0/24, Gig0/1, Gig0/2
10   green                            active    Fa0/1, Fa0/2
20   blue                             active    Fa0/3, Fa0/4
30   pink                             active    Fa0/5, Fa0/6
1002 fddi-default                     active
1003 token-ring-default               active
1004 fddinet-default                  active
1005 trnet-default                    active
Switch#
```

There are 5 main types of VLANs depending on the type of the network they carry:

1. **Default VLAN –**

   When the switch initially starts up, all switch ports become a member of the default VLAN (generally all switches have default VLAN named as **VLAN 1**), which makes them all part of the same broadcast domain. Using default VLAN allows any network device connected to any of the switch port to connect with other devices on other switch ports. One unique feature of Default VLAN is that it can't be rename or delete.

2. **Data VLAN –**

   Data VLAN is used to divide the whole network into 2 groups. One group of users and other group of devices. This VLAN also known as a user VLAN, the data VLAN is used only for user-generated data. This VLAN carrying data only. It is not used for carrying management traffic or voice.

3. **Voice VLAN –**

   Voice VLAN is configured to carry voice traffic. Voice VLANs are mostly given high transmission priority over other types of network traffic. To ensure voice over IP (VoIP) quality (delay of less than 150 milliseconds (ms) across the network), we must have separate voice VLAN as this will preserve bandwidth for other applications.

4. **Management VLAN –**

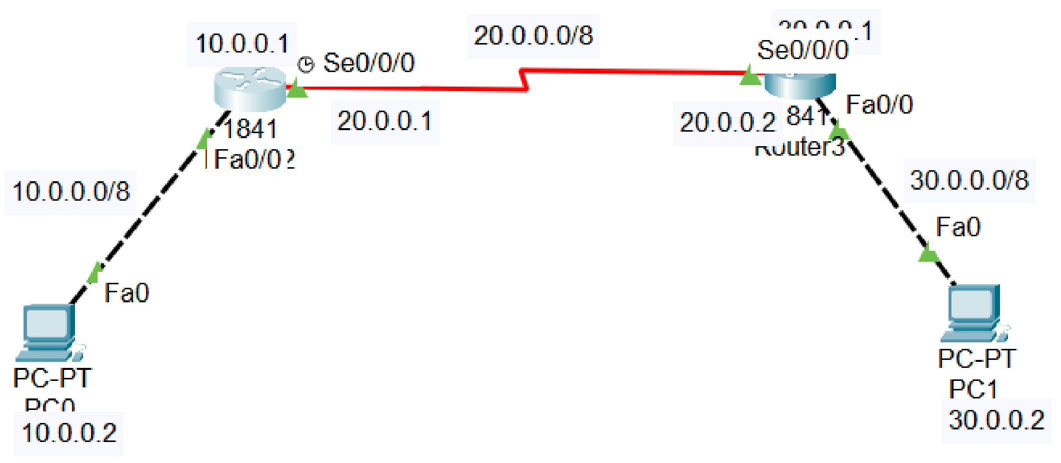
   A management VLAN is configured to access the management capabilities of a switch (traffic like system logging, monitoring). VLAN 1 is the management VLAN by default (VLAN 1 would be a bad choice for the management VLAN). Any of a switch VLAN could be define as the management VLAN if admin as not configured a unique VLAN to serve as the management VLAN. This VLAN ensures that bandwidth for management will be available even when user traffic is high.

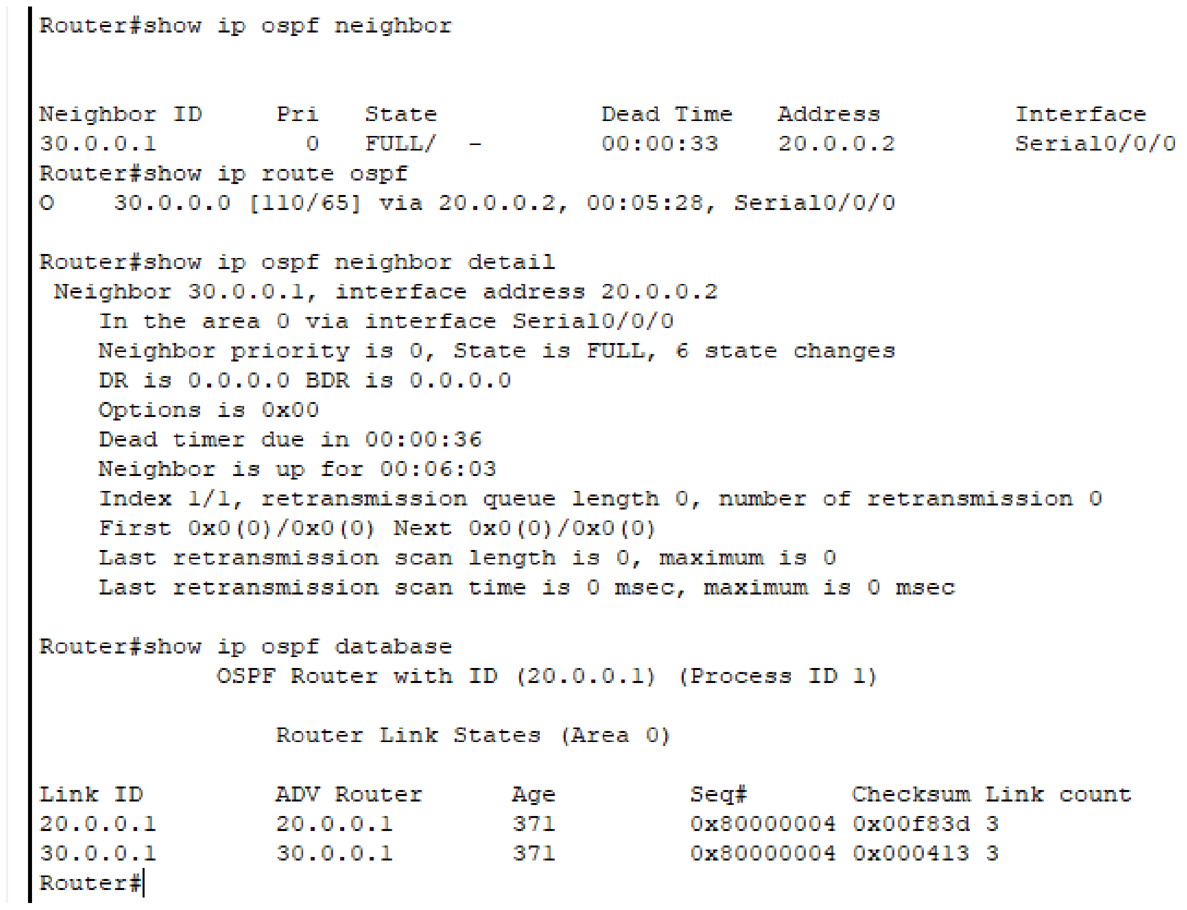
5. **Native VLAN –**

   This VLAN identifies traffic coming from each end of a trunk link. A native VLAN is allocated only to an 802.1Q trunk port. The 802.1Q trunk port places untagged traffic (traffic that does not come from any VLAN) on the native VLAN. It is a best to configure the native VLAN as an unused VLAN.

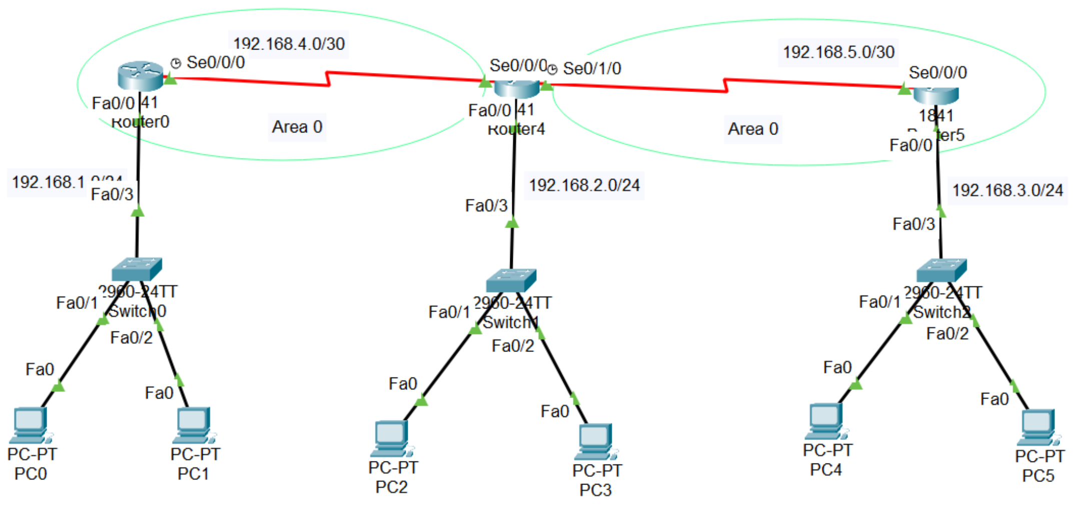
Practical No 2

Aim : OSPF/BGP/NAT Implementations



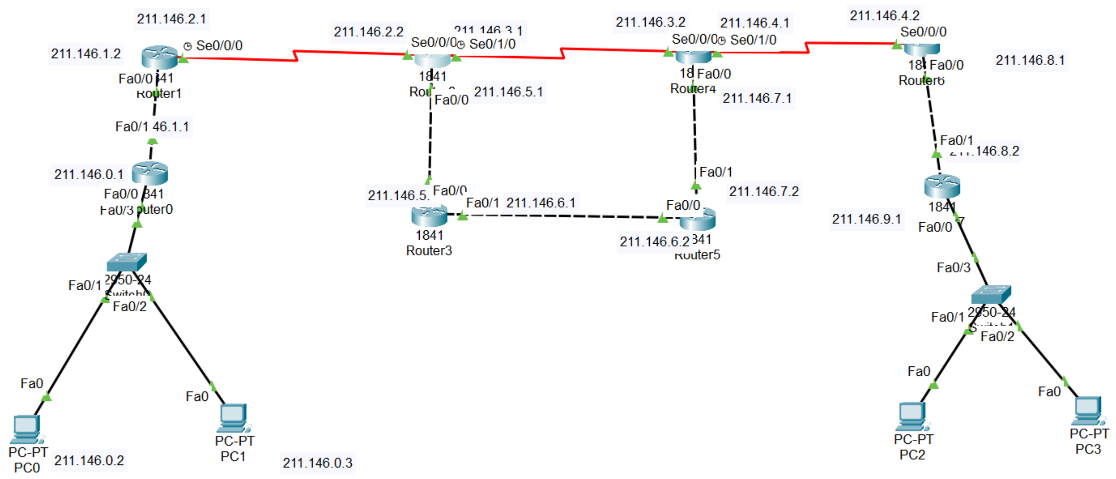

```
Router#show ip ospf neighbor


Neighbor ID     Pri   State           Dead Time   Address         Interface
30.0.0.1          0   FULL/  -        00:00:33    20.0.0.2        Serial0/0/0
Router#show ip route ospf
O    30.0.0.0 [110/65] via 20.0.0.2, 00:05:28, Serial0/0/0

Router#show ip ospf neighbor detail
 Neighbor 30.0.0.1, interface address 20.0.0.2
    In the area 0 via interface Serial0/0/0
    Neighbor priority is 0, State is FULL, 6 state changes
    DR is 0.0.0.0 BDR is 0.0.0.0
    Options is 0x00
    Dead timer due in 00:00:36
    Neighbor is up for 00:06:03
    Index 1/1, retransmission queue length 0, number of retransmission 0
    First 0x0(0)/0x0(0) Next 0x0(0)/0x0(0)
    Last retransmission scan length is 0, maximum is 0
    Last retransmission scan time is 0 msec, maximum is 0 msec

Router#show ip ospf database
            OSPF Router with ID (20.0.0.1) (Process ID 1)

                Router Link States (Area 0)

Link ID         ADV Router      Age         Seq#       Checksum Link count
20.0.0.1        20.0.0.1        371         0x80000004 0x00f83d 3
30.0.0.1        30.0.0.1        371         0x80000004 0x000413 3
Router#
```

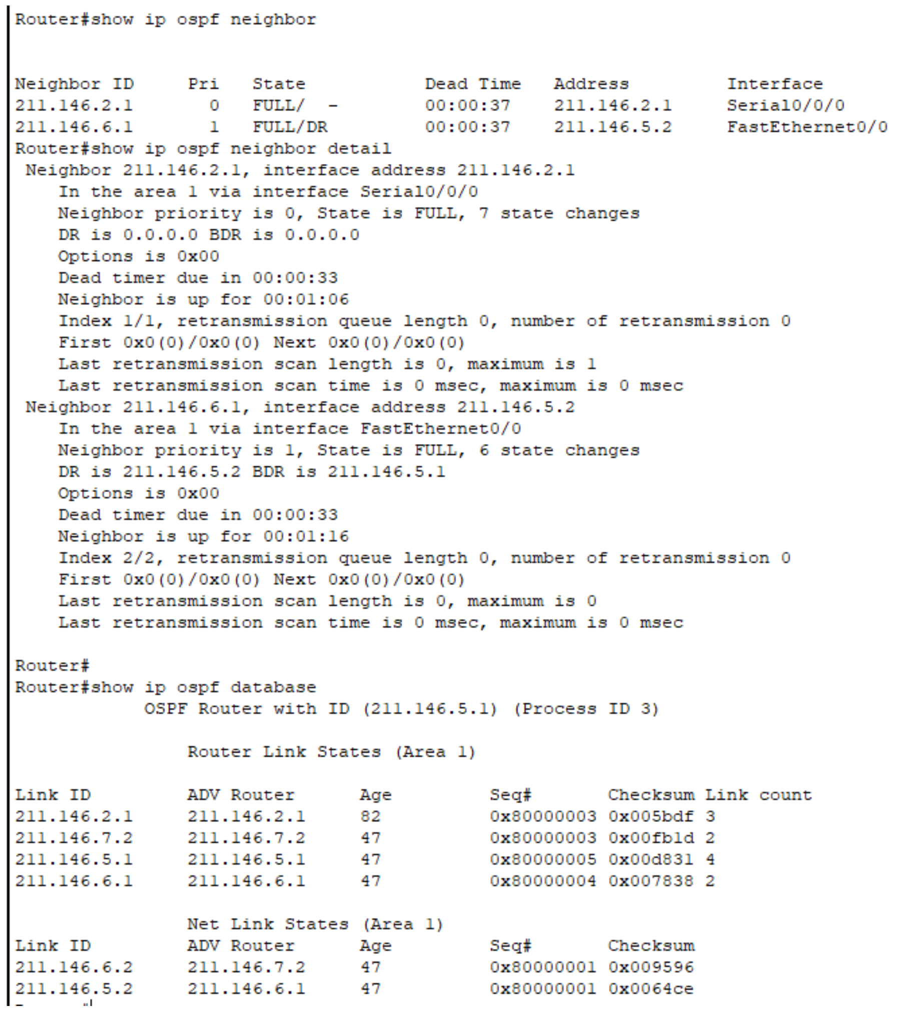
1)



```
Router#show ip ospf neighbor


Neighbor ID     Pri   State           Dead Time   Address         Interface
192.168.5.1       0   FULL/  -        00:00:30    192.168.4.2     Serial0/0/0
Router#show ip route ospf
O    192.168.2.0 [110/65] via 192.168.4.2, 00:04:14, Serial0/0/0
O    192.168.3.0 [110/129] via 192.168.4.2, 00:04:14, Serial0/0/0
O    192.168.5.0 [110/128] via 192.168.4.2, 00:04:14, Serial0/0/0

Router#show ip ospf neighbor detail
 Neighbor 192.168.5.1, interface address 192.168.4.2
    In the area 0 via interface Serial0/0/0
    Neighbor priority is 0, State is FULL, 6 state changes
    DR is 0.0.0.0 BDR is 0.0.0.0
    Options is 0x00
    Dead timer due in 00:00:30
    Neighbor is up for 00:04:39
    Index 1/1, retransmission queue length 0, number of retransmission 0
    First 0x0(0)/0x0(0) Next 0x0(0)/0x0(0)
    Last retransmission scan length is 0, maximum is 0
    Last retransmission scan time is 0 msec, maximum is 0 msec

Router#show ip ospf database
            OSPF Router with ID (192.168.4.1) (Process ID 1)

                Router Link States (Area 0)

Link ID         ADV Router      Age         Seq#       Checksum Link count
192.168.4.1     192.168.4.1     284         0x80000003 0x00de42 3
192.168.5.1     192.168.5.1     284         0x80000005 0x000d22 5
192.168.5.2     192.168.5.2     284         0x80000003 0x0023f4 3
Router#
```

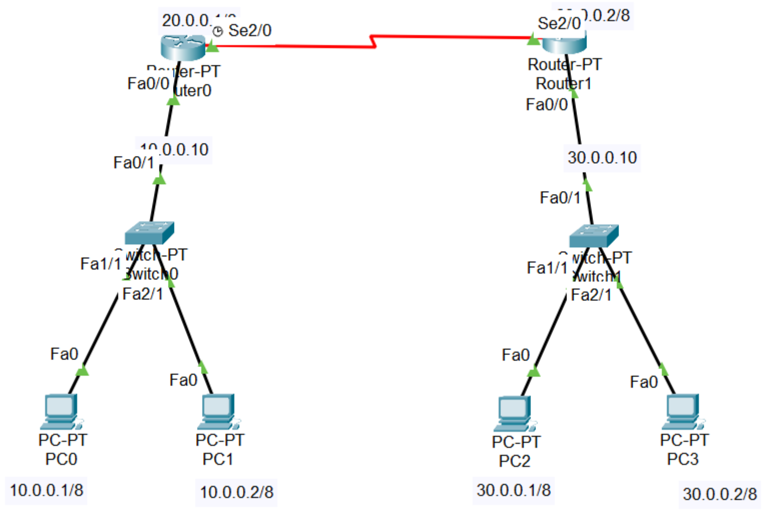
2)

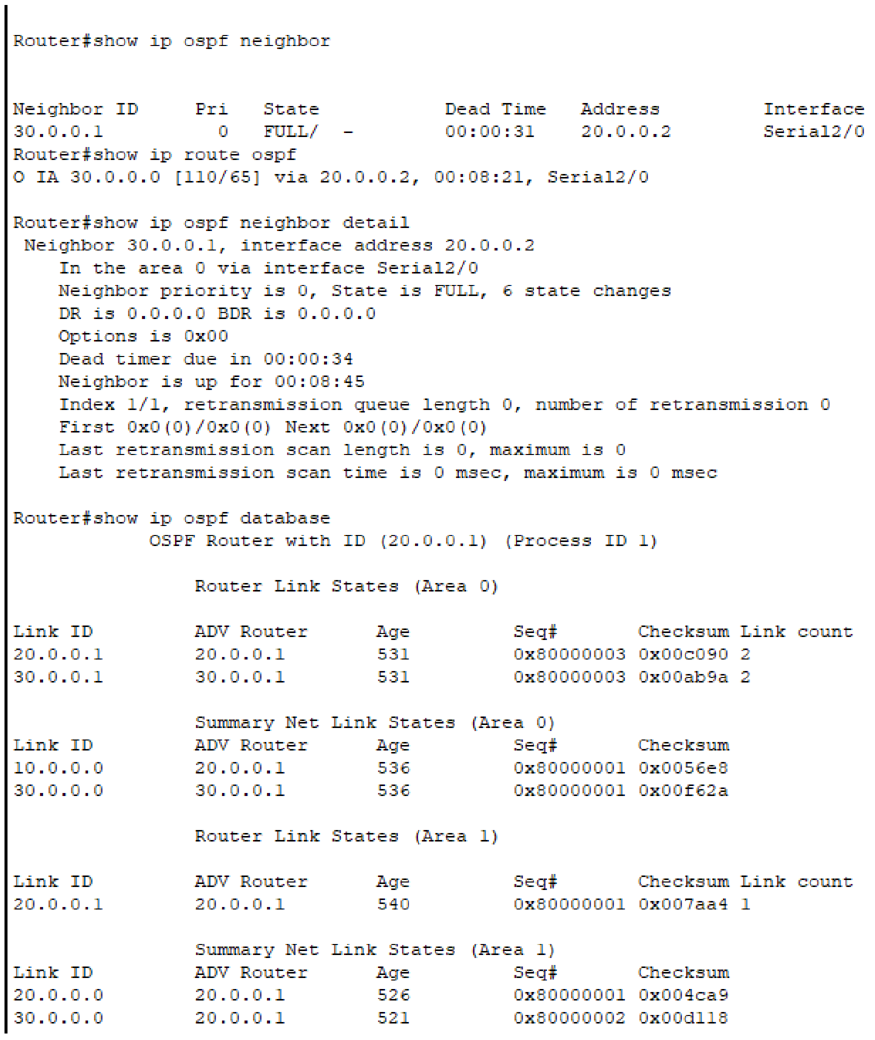




```
Router#show ip ospf neighbor


Neighbor ID     Pri   State           Dead Time   Address         Interface
211.146.2.1       0   FULL/  -        00:00:37    211.146.2.1     Serial0/0/0
211.146.6.1       1   FULL/DR         00:00:37    211.146.5.2     FastEthernet0/0
Router#show ip ospf neighbor detail
 Neighbor 211.146.2.1, interface address 211.146.2.1
    In the area 1 via interface Serial0/0/0
    Neighbor priority is 0, State is FULL, 7 state changes
    DR is 0.0.0.0 BDR is 0.0.0.0
    Options is 0x00
    Dead timer due in 00:00:33
    Neighbor is up for 00:01:06
    Index 1/1, retransmission queue length 0, number of retransmission 0
    First 0x0(0)/0x0(0) Next 0x0(0)/0x0(0)
    Last retransmission scan length is 0, maximum is 1
    Last retransmission scan time is 0 msec, maximum is 0 msec
 Neighbor 211.146.6.1, interface address 211.146.5.2
    In the area 1 via interface FastEthernet0/0
    Neighbor priority is 1, State is FULL, 6 state changes
    DR is 211.146.5.2 BDR is 211.146.5.1
    Options is 0x00
    Dead timer due in 00:00:33
    Neighbor is up for 00:01:16
    Index 2/2, retransmission queue length 0, number of retransmission 0
    First 0x0(0)/0x0(0) Next 0x0(0)/0x0(0)
    Last retransmission scan length is 0, maximum is 0
    Last retransmission scan time is 0 msec, maximum is 0 msec

Router#
Router#show ip ospf database
            OSPF Router with ID (211.146.5.1) (Process ID 3)

                Router Link States (Area 1)

Link ID         ADV Router      Age         Seq#       Checksum Link count
211.146.2.1     211.146.2.1     82          0x80000003 0x005bdf 3
211.146.7.2     211.146.7.2     47          0x80000003 0x00fb1d 2
211.146.5.1     211.146.5.1     47          0x80000005 0x00d831 4
211.146.6.1     211.146.6.1     47          0x80000004 0x007838 2

                Net Link States (Area 1)
Link ID         ADV Router      Age         Seq#       Checksum
211.146.6.2     211.146.7.2     47          0x80000001 0x009596
211.146.5.2     211.146.6.1     47          0x80000001 0x0064ce
```

3)





```
Router#show ip ospf neighbor


Neighbor ID     Pri   State           Dead Time   Address         Interface
30.0.0.1          0   FULL/  -        00:00:31    20.0.0.2        Serial2/0
Router#show ip route ospf
O IA 30.0.0.0 [110/65] via 20.0.0.2, 00:08:21, Serial2/0

Router#show ip ospf neighbor detail
 Neighbor 30.0.0.1, interface address 20.0.0.2
    In the area 0 via interface Serial2/0
    Neighbor priority is 0, State is FULL, 6 state changes
    DR is 0.0.0.0 BDR is 0.0.0.0
    Options is 0x00
    Dead timer due in 00:00:34
    Neighbor is up for 00:08:45
    Index 1/1, retransmission queue length 0, number of retransmission 0
    First 0x0(0)/0x0(0) Next 0x0(0)/0x0(0)
    Last retransmission scan length is 0, maximum is 0
    Last retransmission scan time is 0 msec, maximum is 0 msec

Router#show ip ospf database
            OSPF Router with ID (20.0.0.1) (Process ID 1)

                Router Link States (Area 0)

Link ID         ADV Router      Age         Seq#       Checksum Link count
20.0.0.1        20.0.0.1        531         0x80000003 0x00c090 2
30.0.0.1        30.0.0.1        531         0x80000003 0x00ab9a 2

                Summary Net Link States (Area 0)
Link ID         ADV Router      Age         Seq#       Checksum
10.0.0.0        20.0.0.1        536         0x80000001 0x0056e8
30.0.0.0        30.0.0.1        536         0x80000001 0x00f62a

                Router Link States (Area 1)

Link ID         ADV Router      Age         Seq#       Checksum Link count
20.0.0.1        20.0.0.1        540         0x80000001 0x007aa4 1

                Summary Net Link States (Area 1)
Link ID         ADV Router      Age         Seq#       Checksum
20.0.0.0        20.0.0.1        526         0x80000001 0x004ca9
30.0.0.0        20.0.0.1        521         0x80000002 0x00d118
```

Routing is the process of establishing the routes that data packets must follow to reach the destination. In this process, a routing table is created which contains information regarding routes that data packets follow. Various routing algorithms are used for the purpose of deciding which route an incoming data packet needs to be transmitted on to reach the destination efficiently.

1. Adaptive Algorithms –

These are the algorithms that change their routing decisions whenever network topology or traffic load changes. The changes in routing decisions are reflected in the topology as well as the traffic of the network. Also known as dynamic routing, these make use of dynamic information such as current topology, load, delay, etc. to select routes. Optimization parameters are distance, number of hops, and estimated transit time.

Further, these are classified as follows:

- **(a) Isolated –** In this method each, node makes its routing decisions using the information it has without seeking information from other nodes. The sending nodes don't have information about the status of a particular link. The disadvantage is that packets may be sent through a congested network which may result in delay. Examples: Hot potato routing, backward learning.
- **(b) Centralized –** In this method, a centralized node has entire information about the network and makes all the routing decisions. The advantage of this is only one node is required to keep the information of the entire network and the disadvantage is that if the central node goes down the entire network is done. The link state algorithm is referred to as a centralized algorithm since it is aware of the cost of each link in the network
- **(c) Distributed –** In this method, the node receives information from its neighbors and then takes the decision about routing the packets. A disadvantage is that the packet may be delayed if there is a change in between intervals in which it receives information and sends packets. It is also known as a decentralized algorithm as it computes the least-cost path between source and destination

2. Non-Adaptive Algorithms –

These are the algorithms that do not change their routing decisions once they have been selected. This is also known as static routing as a route to be taken is computed in advance and downloaded to routers when a router is booted.

Further, these are classified as follows:

- **(a) Flooding –** This adapts the technique in which every incoming packet is sent on every outgoing line except from which it arrived. One problem with this is that packets may go in a loop and as a result of which a node may receive duplicate packets. These problems can be overcome with the help of sequence numbers, hop count, and spanning trees.

- **(b) Random walk –** In this method, packets are sent host by host or node by node to one of its neighbors randomly. This is a highly robust method that is usually implemented by sending packets onto the link which is least queued.

The OSPF (Open Shortest Path First) protocol is one of a family of IP Routing protocols, and is an Interior Gateway Protocol (IGP) for the Internet, used to distribute IP routing information throughout a single Autonomous System (AS) in an IP network.

The OSPF protocol is a link-state routing protocol, which means that the routers exchange topology information with their nearest neighbors. The topology information is flooded throughout the AS, so that every router within the AS has a complete picture of the topology of the AS. This picture is then used to calculate end-to-end paths through the AS, normally using a variant of the Dijkstra algorithm. Therefore, in a link-state routing protocol, the next hop address to which data is forwarded is determined by choosing the best end-to-end path to the eventual destination.

The main advantage of a link state routing protocol like OSPF is that the complete knowledge of topology allows routers to calculate routes that satisfy particular criteria. This can be useful for traffic engineering purposes, where routes can be constrained to meet particular quality of service requirements. The main disadvantage of a link state routing protocol is that it does not scale well as more routers are added to the routing domain. Increasing the number of routers increases the size and frequency of the topology updates, and also the length of time it takes to calculate end-to-end routes. This lack of scalability means that a link state routing protocol is unsuitable for routing across the Internet at large, which is the reason why IGPs only route traffic within a single AS.

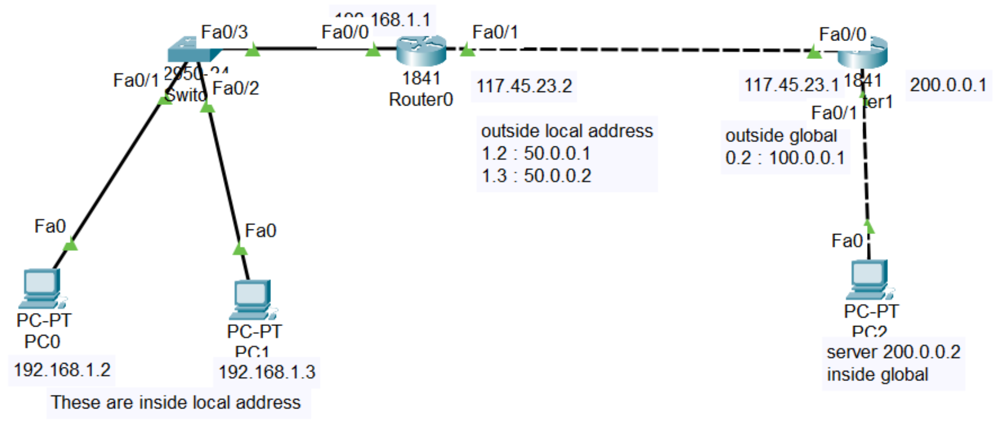
Each OSPF router distributes information about its local state (usable interfaces and reachable neighbors, and the cost of using each interface) to other routers using a Link State Advertisement (LSA) message. Each router uses the received messages to build up an identical database that describes the topology of the AS.

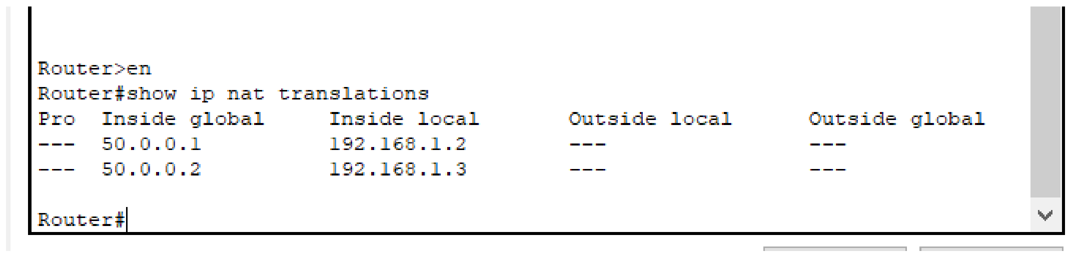
From this database, each router calculates its own routing table using a Shortest Path First (SPF) or Dijkstra algorithm. This routing table contains all the destinations the routing protocol knows about, associated with a next hop IP address and outgoing interface.

- The protocol recalculates routes when network topology changes, using the Dijkstra algorithm, and minimises the routing protocol traffic that it generates.
- It provides support for multiple paths of equal cost.
- It provides a multi-level hierarchy (two-level for OSPF) called "area routing," so that information about the topology within a defined area of the AS is hidden from routers outside this area. This enables an additional level of routing protection and a reduction in routing protocol traffic.
- All protocol exchanges can be authenticated so that only trusted routers can join in the routing exchanges for the AS.

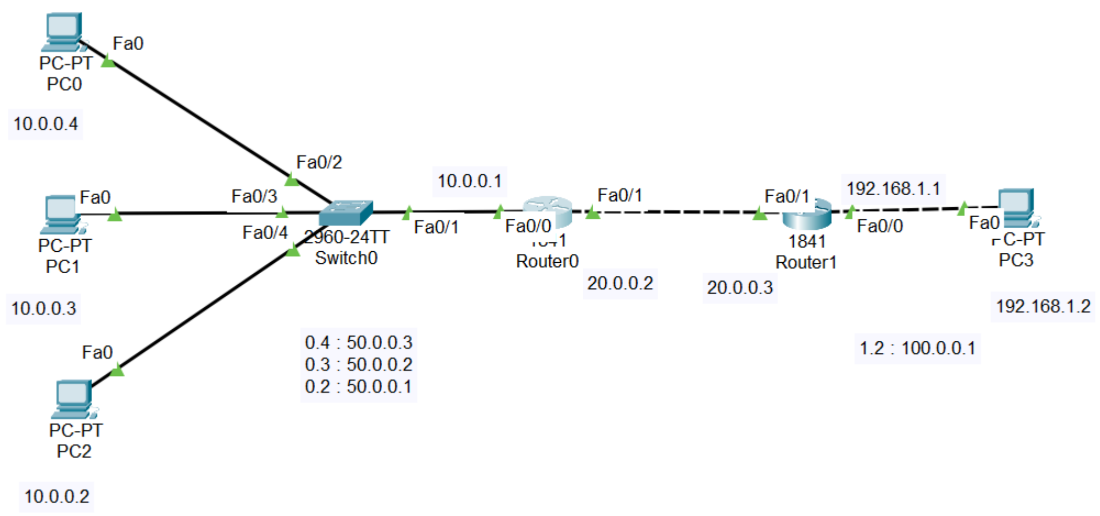
Aim : Implementing NAT

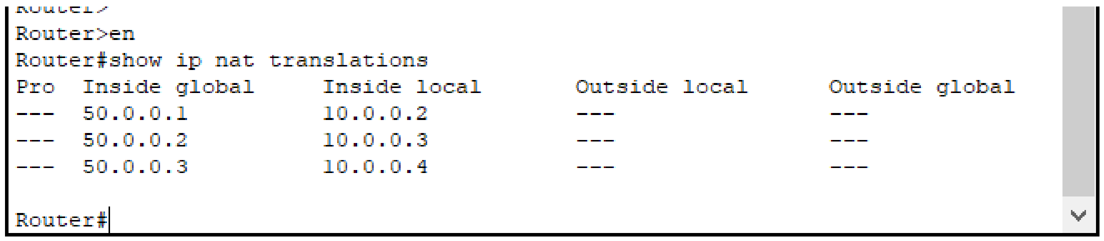
Q1)




```
Router>en
Router#show ip nat translations
Pro  Inside global      Inside local       Outside local      Outside global
---  50.0.0.1           192.168.1.2        ---                ---
---  50.0.0.2           192.168.1.3        ---                ---

Router#
```

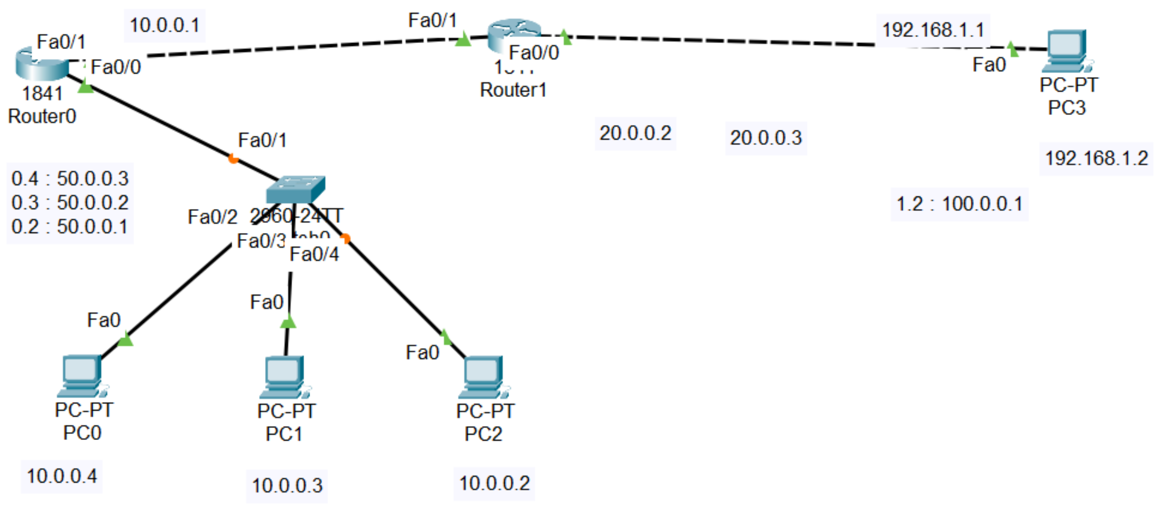
Q2)

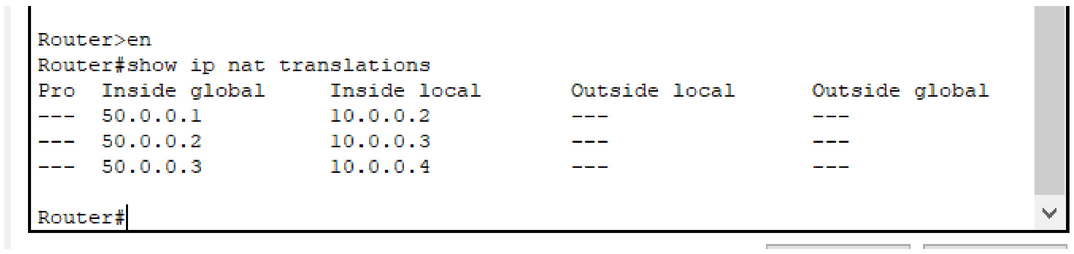




```
Router>
Router>en
Router#show ip nat translations
Pro  Inside global      Inside local       Outside local      Outside global
---  50.0.0.1           10.0.0.2           ---                ---
---  50.0.0.2           10.0.0.3           ---                ---
---  50.0.0.3           10.0.0.4           ---                ---

Router#
```

Q3)





```
Router>en
Router#show ip nat translations
Pro  Inside global      Inside local       Outside local      Outside global
---  50.0.0.1           10.0.0.2           ---                ---
---  50.0.0.2           10.0.0.3           ---                ---
---  50.0.0.3           10.0.0.4           ---                ---

Router#
```

Network Address Translation (NAT) is used to convert a private IP address into a public IP address and also a public IP address into private IP address. We use NAT due to the shortage of IP addresses.

IP Version 4 is a 32- bit address and it has almost 4.3 billion IP addresses, but the population of the world is much higher and it is approximately 7.8 billion as of the latest reports of June 2020.

Most of them use multiple gadgets and devices like smartphones, laptops, tablets, and many more for accessing the internet for various needs. All these devices need an IP address.

So, 4.3 billion of IP addresses is not sufficient for more than 7.8 billion of people using , that is the reason why we are using NAT. NAT is used to convert a private IP address onto public and public IP address into private.

Types of NAT

The types of Network Address Translation (NAT) are explained below −

Static NAT

It is otherwise called balanced NAT. In this sort of NAT, just the IP addresses and the header checksum are changed among the general organization addresses.

These are actualized for interconnection of two particular IP networks having contrary tending to.

STATIC NAT is ONE to ONE NAT. This means in the local area network side and outside network side we need an equal number of IP addresses to translate. Due to that reason this is a bit expensive because we need more Public IP addresses.

Keep in mind Public IP addresses are always expensive. In this example 192.168.4.1 always translates to 225.20.120.11.

Dynamic NAT

In this kind of NAT, planning of IP from an unregistered private organization is finished with the single IP address of the enrolled network from the class of enlisted IP addresses.

Dynamic NAT means we have a Pool of IP addresses in the outside list. So which IP addresses are Free in the outside list which will associate to LAN side IP first.

This example 192.168.4.1 from Local will check which Public IP address is from the pool.

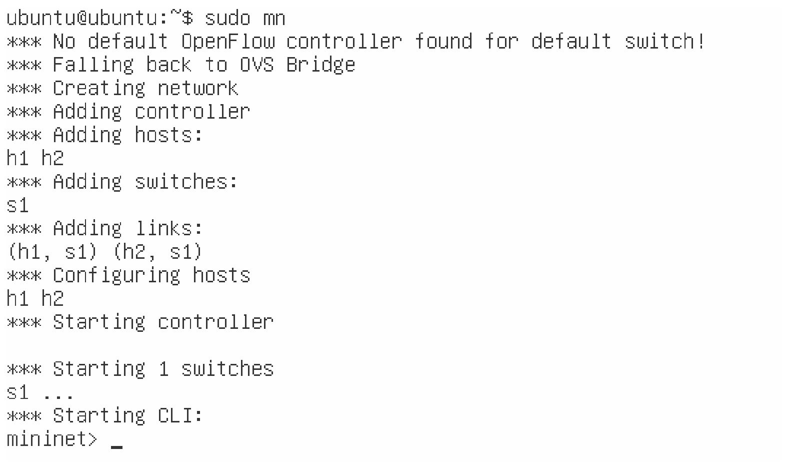
If First IP free then it translates to First IP. If the first IP is already associated with any other local side IP, it will check the next available IP.

Suppose there is no IP available from the Public IP pool for translating Private IP addresses. Then, that traffic will drop by Router or Firewall.

Aim : Creating SDN networks using Mininet

Q1) Install and create the environment to execute the following

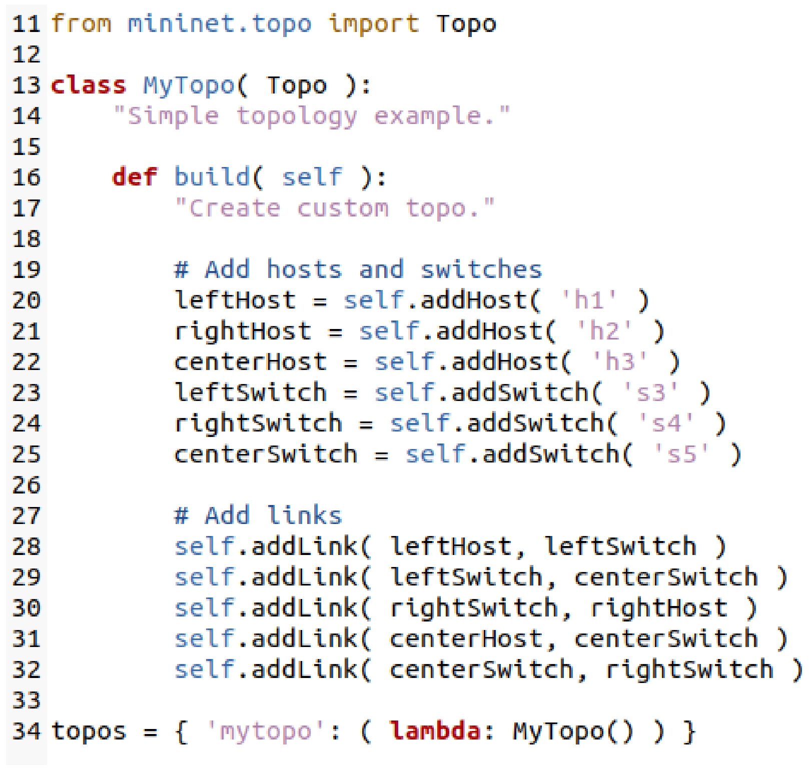
a. Display node



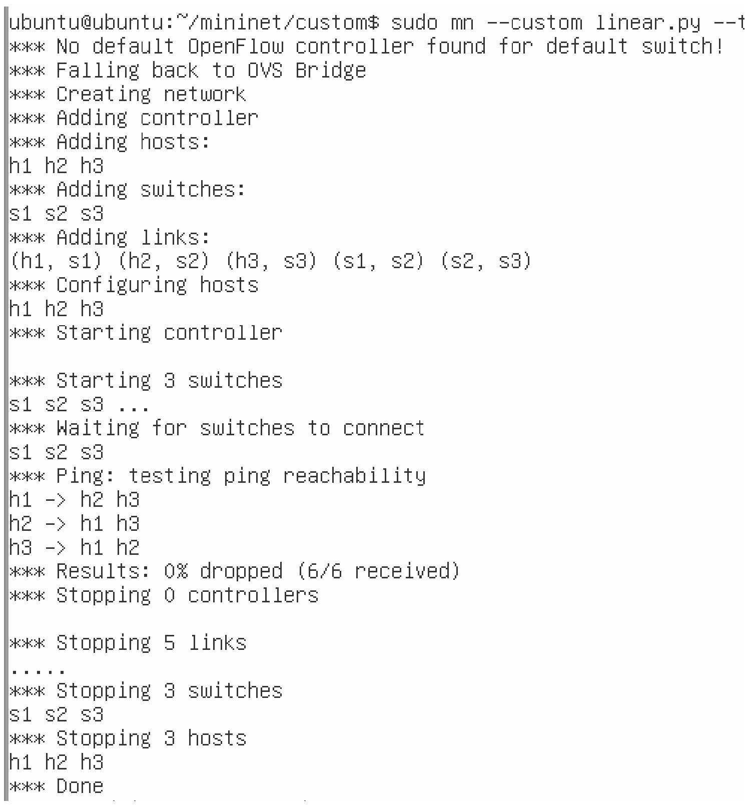
```
ubuntu@ubuntu:~$ sudo mn
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> _
```

b. Display topology (all 3 type-simple linear and tree)

Linear

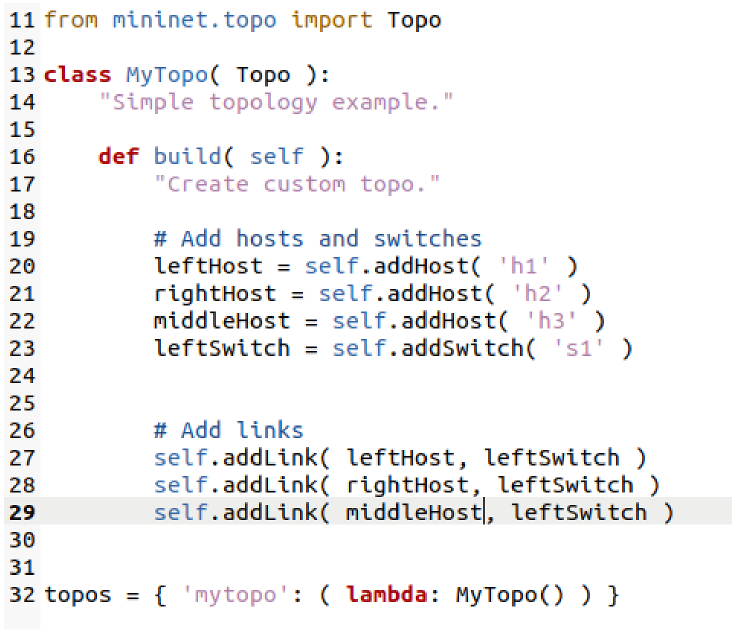


```python
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        centerHost = self.addHost( 'h3' )
        leftSwitch = self.addSwitch( 's3' )
        rightSwitch = self.addSwitch( 's4' )
        centerSwitch = self.addSwitch( 's5' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, centerSwitch )
        self.addLink( rightSwitch, rightHost )
        self.addLink( centerHost, centerSwitch )
        self.addLink( centerSwitch, rightSwitch )

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

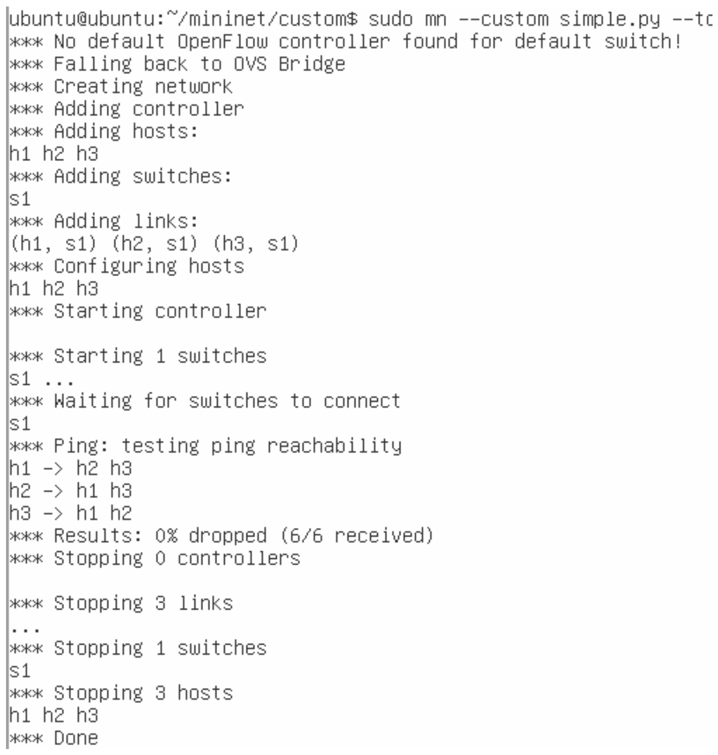


```
ubuntu@ubuntu:~/mininet/custom$ sudo mn --custom linear.py --t
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3
*** Starting controller

*** Starting 3 switches
s1 s2 s3 ...
*** Waiting for switches to connect
s1 s2 s3
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
*** Stopping 0 controllers

*** Stopping 5 links
.....
*** Stopping 3 switches
s1 s2 s3
*** Stopping 3 hosts
h1 h2 h3
*** Done
```

Simple

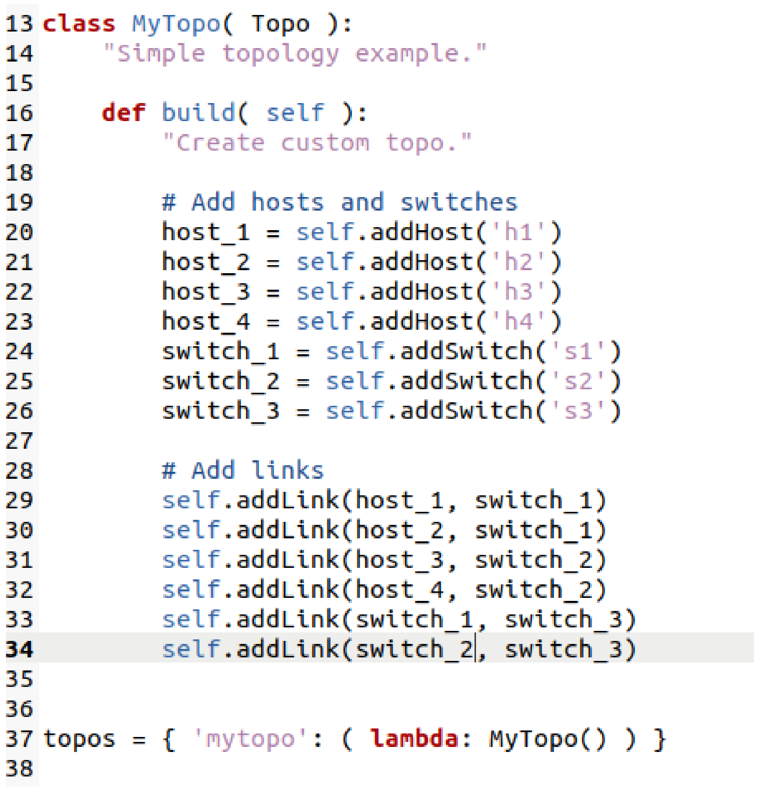


```python
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts and switches
        leftHost = self.addHost( 'h1' )
        rightHost = self.addHost( 'h2' )
        middleHost = self.addHost( 'h3' )
        leftSwitch = self.addSwitch( 's1' )


        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( rightHost, leftSwitch )
        self.addLink( middleHost, leftSwitch )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

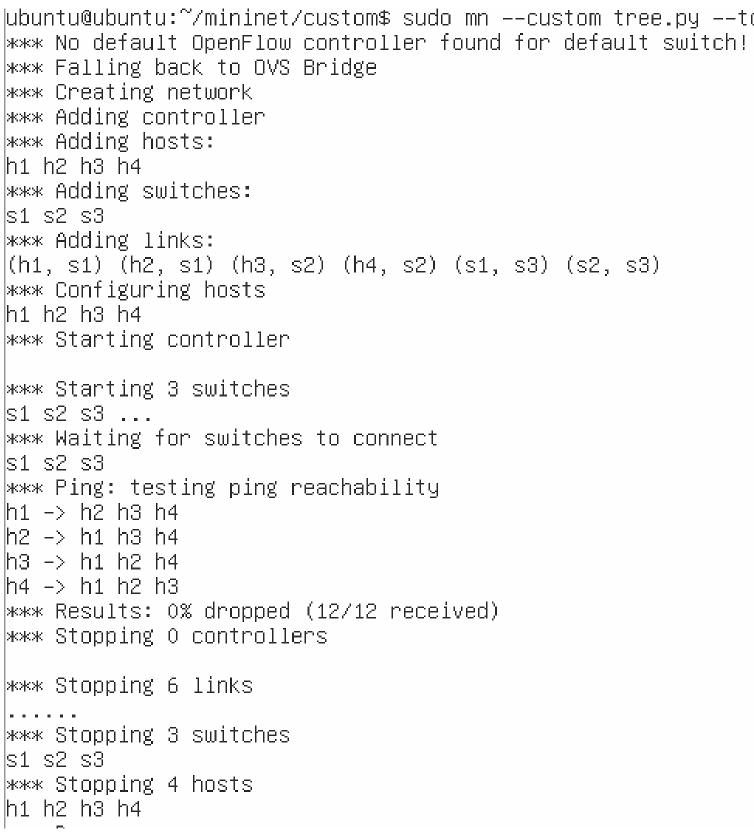


```
ubuntu@ubuntu:~/mininet/custom$ sudo mn --custom simple.py --tc
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller

*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
*** Stopping 0 controllers

*** Stopping 3 links
...
*** Stopping 1 switches
s1
*** Stopping 3 hosts
h1 h2 h3
*** Done
```

Tree

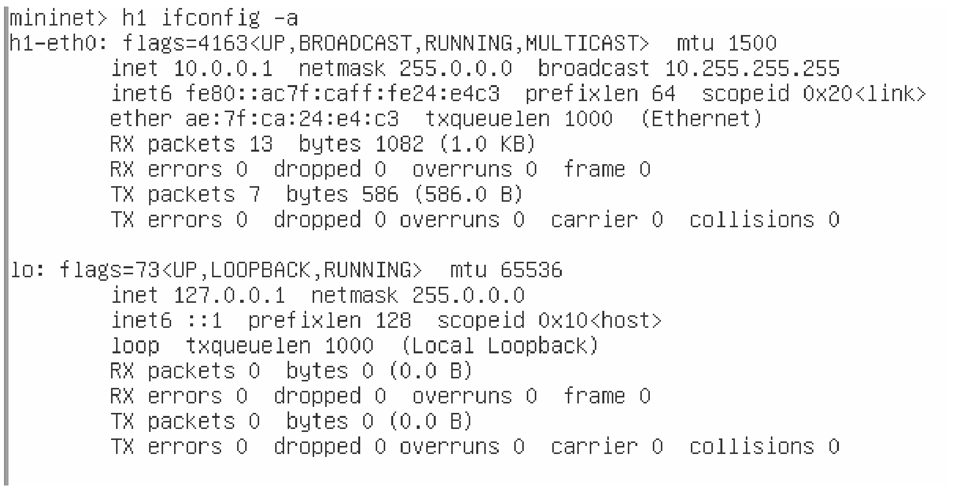


```python
class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts and switches
        host_1 = self.addHost('h1')
        host_2 = self.addHost('h2')
        host_3 = self.addHost('h3')
        host_4 = self.addHost('h4')
        switch_1 = self.addSwitch('s1')
        switch_2 = self.addSwitch('s2')
        switch_3 = self.addSwitch('s3')

        # Add links
        self.addLink(host_1, switch_1)
        self.addLink(host_2, switch_1)
        self.addLink(host_3, switch_2)
        self.addLink(host_4, switch_2)
        self.addLink(switch_1, switch_3)
        self.addLink(switch_2, switch_3)


topos = { 'mytopo': ( lambda: MyTopo() ) }

```



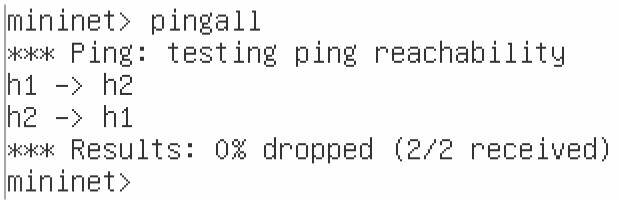
```
ubuntu@ubuntu:~/mininet/custom$ sudo mn --custom tree.py --t
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (s1, s3) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller

*** Starting 3 switches
s1 s2 s3 ...
*** Waiting for switches to connect
s1 s2 s3
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
*** Stopping 0 controllers

*** Stopping 6 links
......
*** Stopping 3 switches
s1 s2 s3
*** Stopping 4 hosts
h1 h2 h3 h4
```

c. Show the network interface



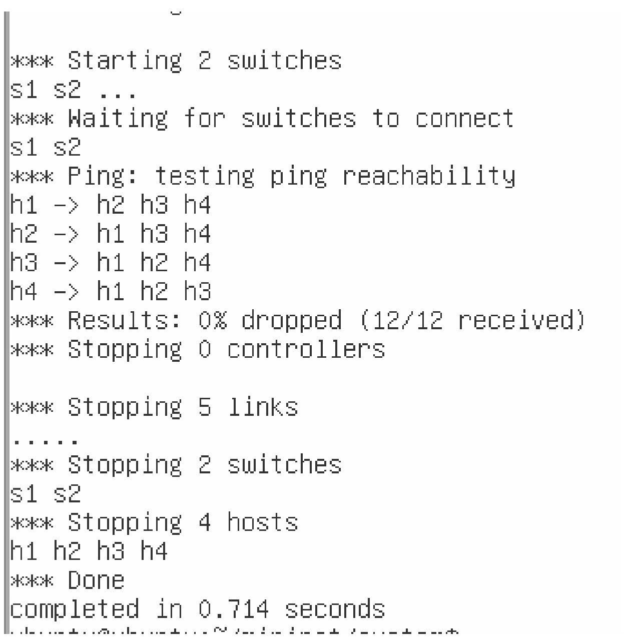
```
mininet> h1 ifconfig -a
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.1  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::ac7f:caff:fe24:e4c3  prefixlen 64  scopeid 0x20<link>
        ether ae:7f:ca:24:e4:c3  txqueuelen 1000  (Ethernet)
        RX packets 13  bytes 1082 (1.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7  bytes 586 (586.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

d. Test the connectivity



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

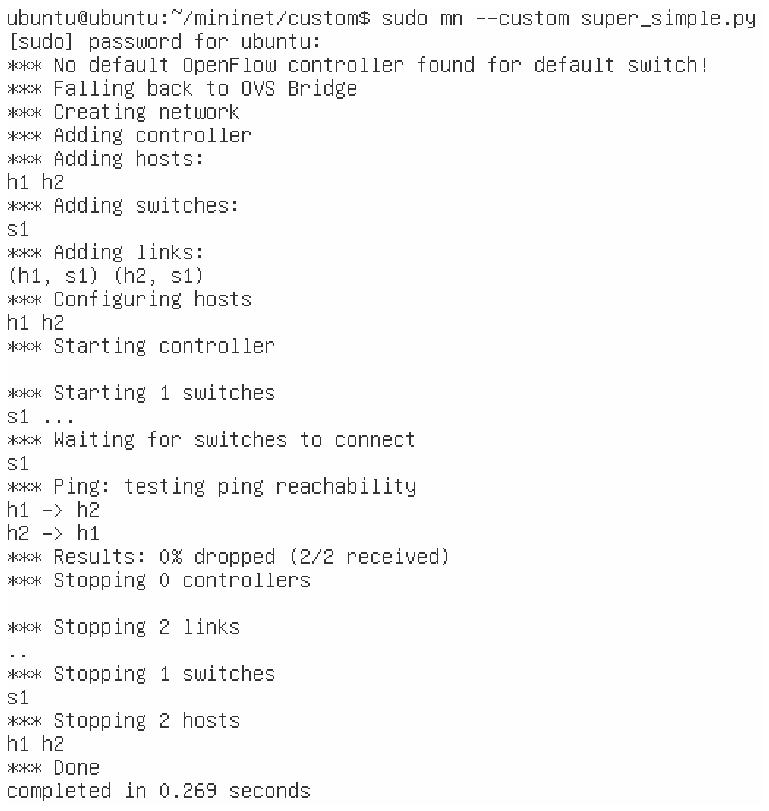
e. Execute the default custom topology



```
*** Starting 2 switches
s1 s2 ...
*** Waiting for switches to connect
s1 s2
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
*** Stopping 0 controllers

*** Stopping 5 links
.....
*** Stopping 2 switches
s1 s2
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 0.714 seconds
```

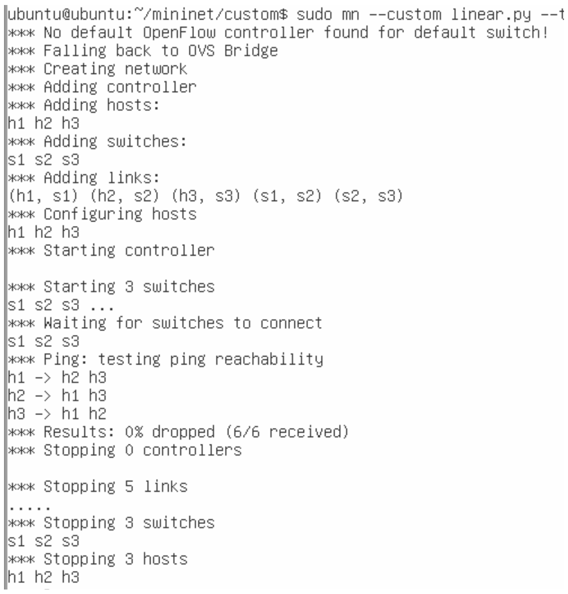
Q2) Implement the following custom topologies using mininet.

a. Simple network with 2 hosts and 1 switch.



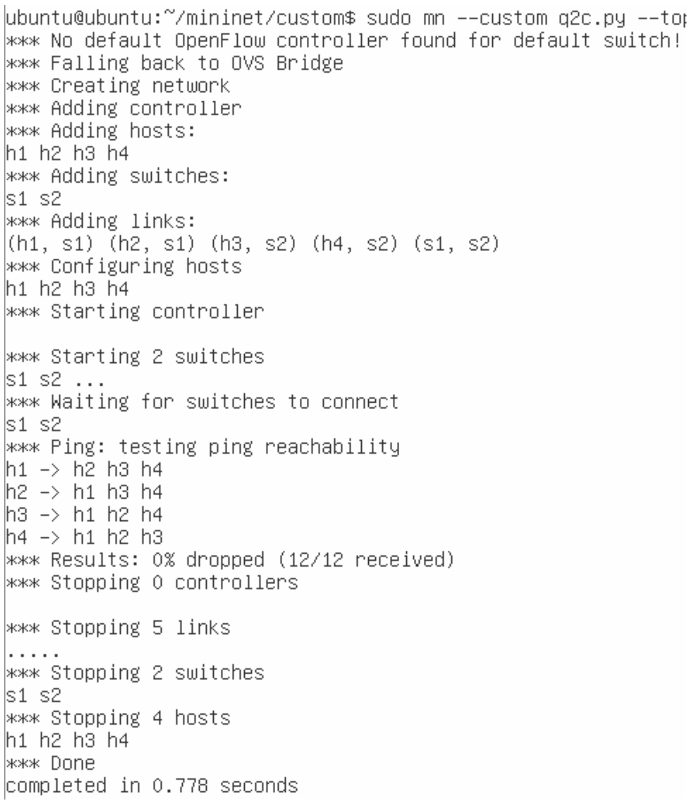
```
ubuntu@ubuntu:~/mininet/custom$ sudo mn --custom super_simple.py
[sudo] password for ubuntu:
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller

*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 0 controllers

*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 0.269 seconds
```

b. Design a simple linear network with 3 switches and 3 hosts.



```
ubuntu@ubuntu:~/mininet/custom$ sudo mn --custom linear.py --t
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3
*** Starting controller

*** Starting 3 switches
s1 s2 s3 ...
*** Waiting for switches to connect
s1 s2 s3
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
*** Stopping 0 controllers

*** Stopping 5 links
.....
*** Stopping 3 switches
s1 s2 s3
*** Stopping 3 hosts
h1 h2 h3
```

c. Design the following



```
ubuntu@ubuntu:~/mininet/custom$ sudo mn --custom q2c.py --to
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller

*** Starting 2 switches
s1 s2 ...
*** Waiting for switches to connect
s1 s2
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
*** Stopping 0 controllers

*** Stopping 5 links
.....
*** Stopping 2 switches
s1 s2
*** Stopping 4 hosts
h1 h2 h3 h4
*** Done
completed in 0.778 seconds
```

SDN

Software-Defined Networking (SDN) is an approach to networking that uses software-based controllers or application programming interfaces (APIs) to communicate with underlying hardware infrastructure and direct traffic on a network.
This model differs from that of traditional networks, which use dedicated hardware devices (i.e., routers and switches) to control network traffic. SDN can create and control a virtual network – or control a traditional hardware – via software.

Why SDN

Typically, networks have been defined and governed by their tangible properties—the hardware and wiring that make the physical connections between points. Software-Defined Networking (SDN)—a burgeoning network architecture—is turning that approach on its head. SDN separates the control of the network from the hardware. It uses software applications to program your network intelligently through centralized control. This means the underlying hardware and associated technologies are still there, but they are programmed centrally. As a result, you can consistently and holistically manage your entire network with utmost flexibility and speed.

Mininet is a software emulator for prototyping a large network on a single machine.
Mininet can be used to quickly create a realistic virtual network running actual kernel, switch and software application code on a personal computer. Mininet allows the user to quickly create,
interact with, customize and share a software-defined network (SDN) prototype to simulate a network topology that uses Openflow switches.

Commands:

- Display Mininet CLI commands:
  - `mininet> help`
- Display nodes:
  - `mininet> nodes`
- Display links:
  - `mininet> net`
- Dump information about all nodes:
  - `mininet> dump`
- Show network information
  - `h1 ifconfig -a`

Installation:

```
sudo mn -h
```

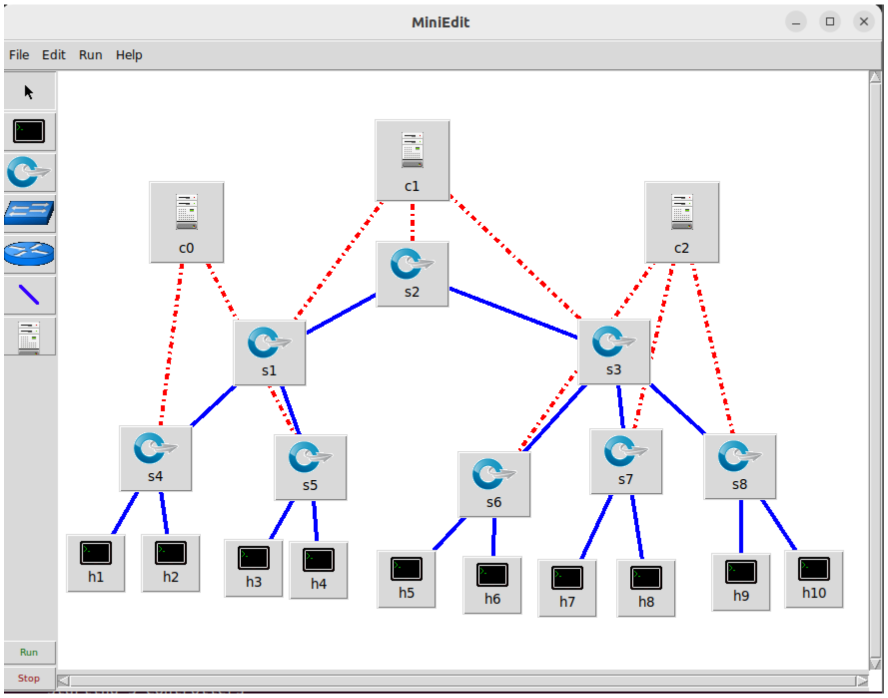
```
git clone https://github.com/mininet/mininet
```

Three topologies :

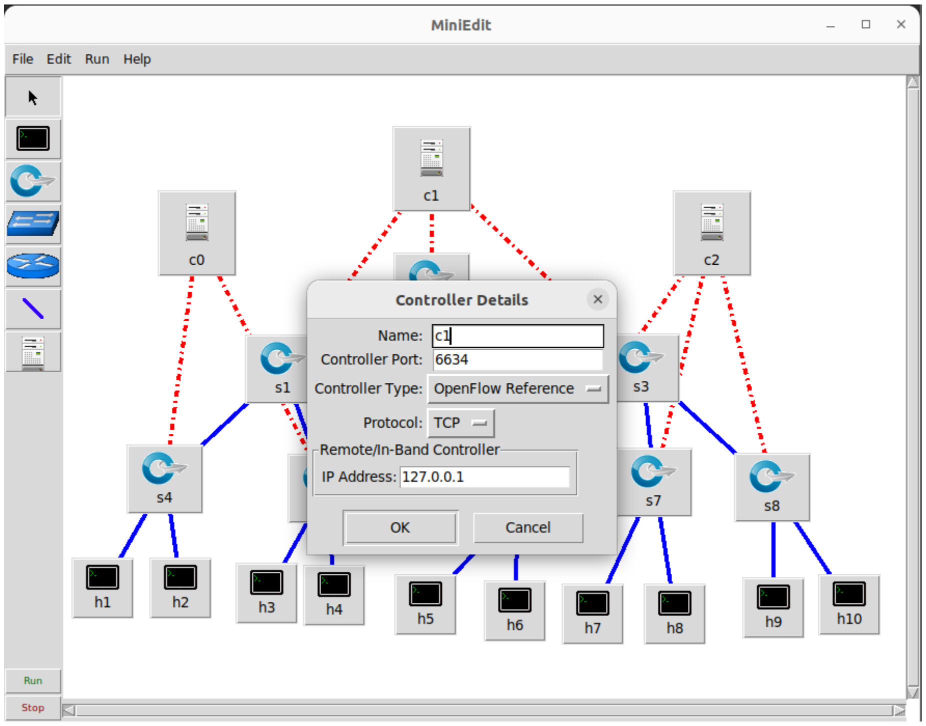
1. Simple
2. Linear
3. Tree

Aim : To create and save topologies in Miniedit

Step 1 : Open miniedit GUI with sudo privileges, and create the following topologies

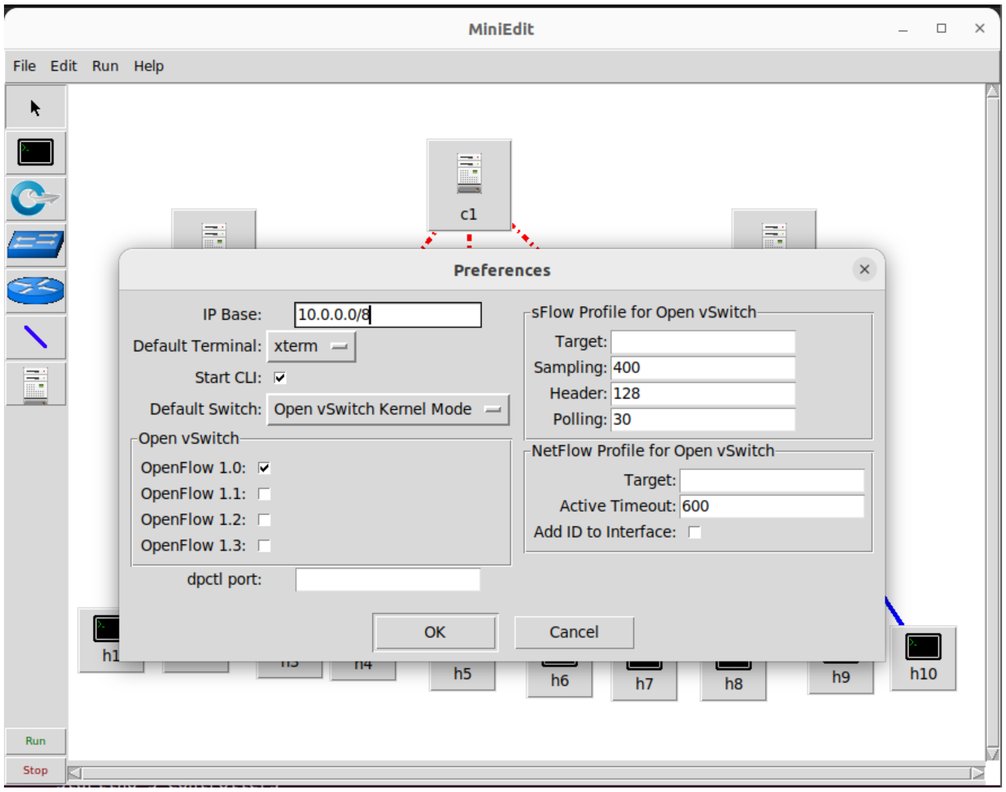


Step 2 : Configure the following controllers placed in the topology



Name: c1
Controller Port: 6634
Controller Type: OpenFlow Reference
Protocol: TCP
Remote/In-Band Controller
IP Address: 127.0.0.1

Step 3 : Under Edit > Preferences select the checkbox start cli



Preferences

IP Base: 10.0.0.0/8
Default Terminal: xterm
Start CLI: ☑
Default Switch: Open vSwitch Kernel Mode

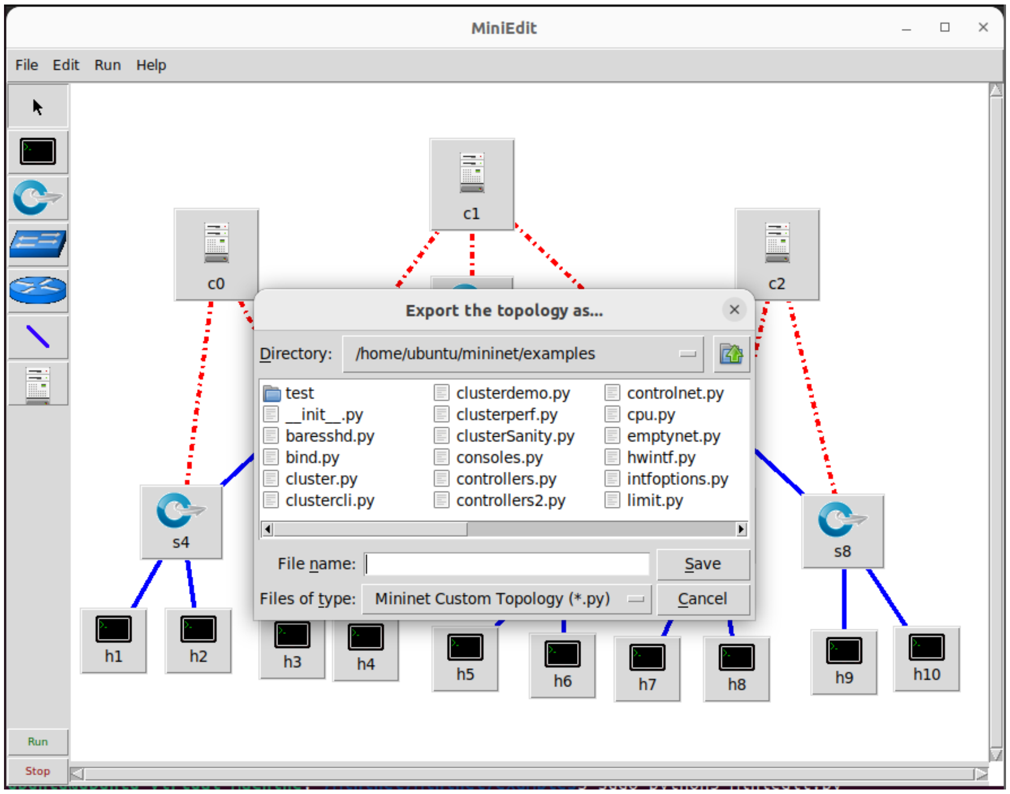
Open vSwitch
OpenFlow 1.0: ☑
OpenFlow 1.1: ☐
OpenFlow 1.2: ☐
OpenFlow 1.3: ☐

dpctl port:

sFlow Profile for Open vSwitch
Target:
Sampling: 400
Header: 128
Polling: 30

NetFlow Profile for Open vSwitch
Target:
Active Timeout: 600
Add ID to Interface: ☐

Step 4 : Save the following configuration under File as .mn file



Export the topology as...

Directory: /home/ubuntu/mininet/examples

test	clusterdemo.py	controlnet.py
__init__.py	clusterperf.py	cpu.py
baresshd.py	clusterSanity.py	emptynet.py
bind.py	consoles.py	hwintf.py
cluster.py	controllers.py	intfoptions.py
clustercli.py	controllers2.py	limit.py

File name:
Files of type: Mininet Custom Topology (*.py)

Write Up:

Mininet is a virtual testbed enabling the development and testing of network tools and protocols. With a single command, Mininet can create a realistic virtual network on any type of machine (Virtual Machine (VM), cloud-hosted, or native). Therefore, it provides an inexpensive solution and streamlined development running in line with production networks

Mininet offers the following features:

• Fast prototyping for new networking protocols.

• Simplified testing for complex topologies without the need of buying expensive hardware.

• Realistic execution as it runs real code on the Unix and Linux kernels.

• Open source environment backed by a large community contributing extensive documentation.

Mininet is useful for development, teaching, and research as it is easy to customize and interact with it through the CLI or the GUI. Mininet was originally designed to experiment with OpenFlow and Software-Defined Networking (SDN). This lab, however, only focuses on emulating a simple network environment without SDN-based devices.

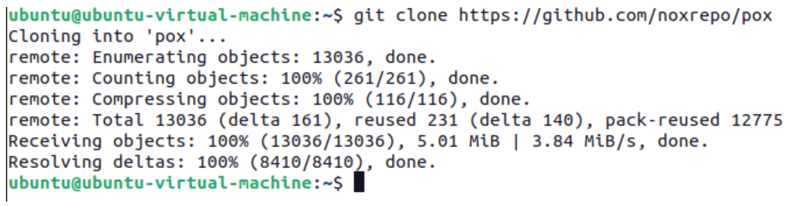
Mininet’s logical nodes can be connected into networks. These nodes are sometimes called containers, or more accurately, network namespaces. Containers consume sufficiently fewer resources that networks of over a thousand nodes have created, running on a single laptop. A Mininet container is a process (or group of processes) that no longer has access to all the host system’s native network interfaces. Containers are then assigned virtual Ethernet interfaces, which are connected to other containers through a virtual switch4. Mininet connects a host and a switch using a virtual Ethernet (veth) link. The veth link is analogous to a wire connecting two virtual interfaces, as illustrated below.

MiniEdit is an experimental tool created to demonstrate how Mininet can be extended. To show how to use MiniEdit to create and run network simulations, we will work through a tutorial that demonstrates how to use MiniEdit to build a network, configure network elements, save the topology, and run the simulation.

Aim : Implementing Firewall using pox controller

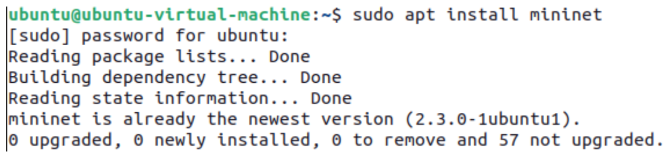
Steps :

Clone the repo



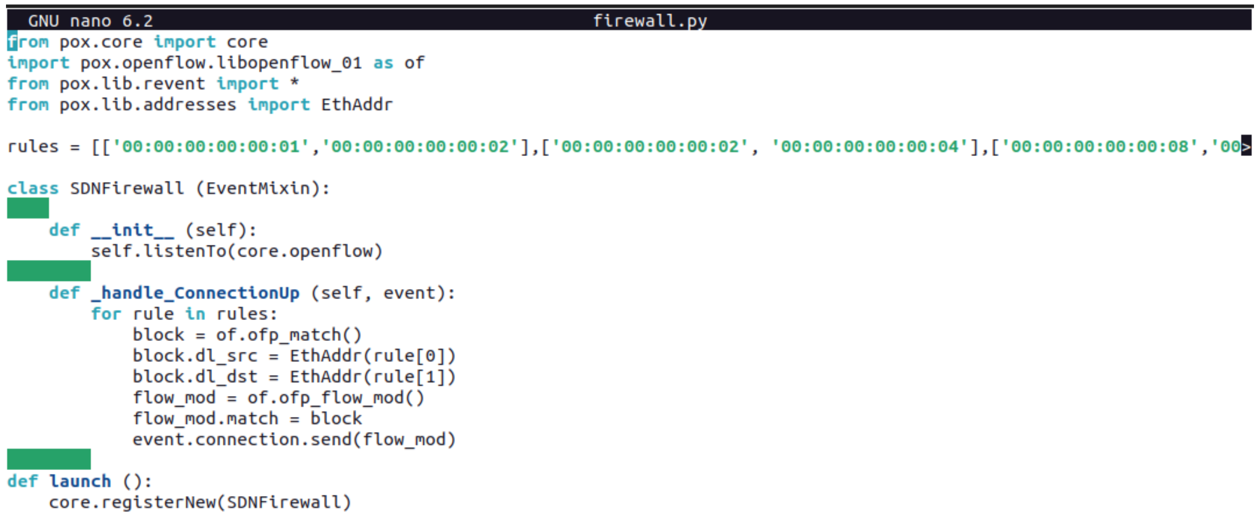
```
ubuntu@ubuntu-virtual-machine:~$ git clone https://github.com/noxrepo/pox
Cloning into 'pox'...
remote: Enumerating objects: 13036, done.
remote: Counting objects: 100% (261/261), done.
remote: Compressing objects: 100% (116/116), done.
remote: Total 13036 (delta 161), reused 231 (delta 140), pack-reused 12775
Receiving objects: 100% (13036/13036), 5.01 MiB | 3.84 MiB/s, done.
Resolving deltas: 100% (8410/8410), done.
ubuntu@ubuntu-virtual-machine:~$
```

Install mininet



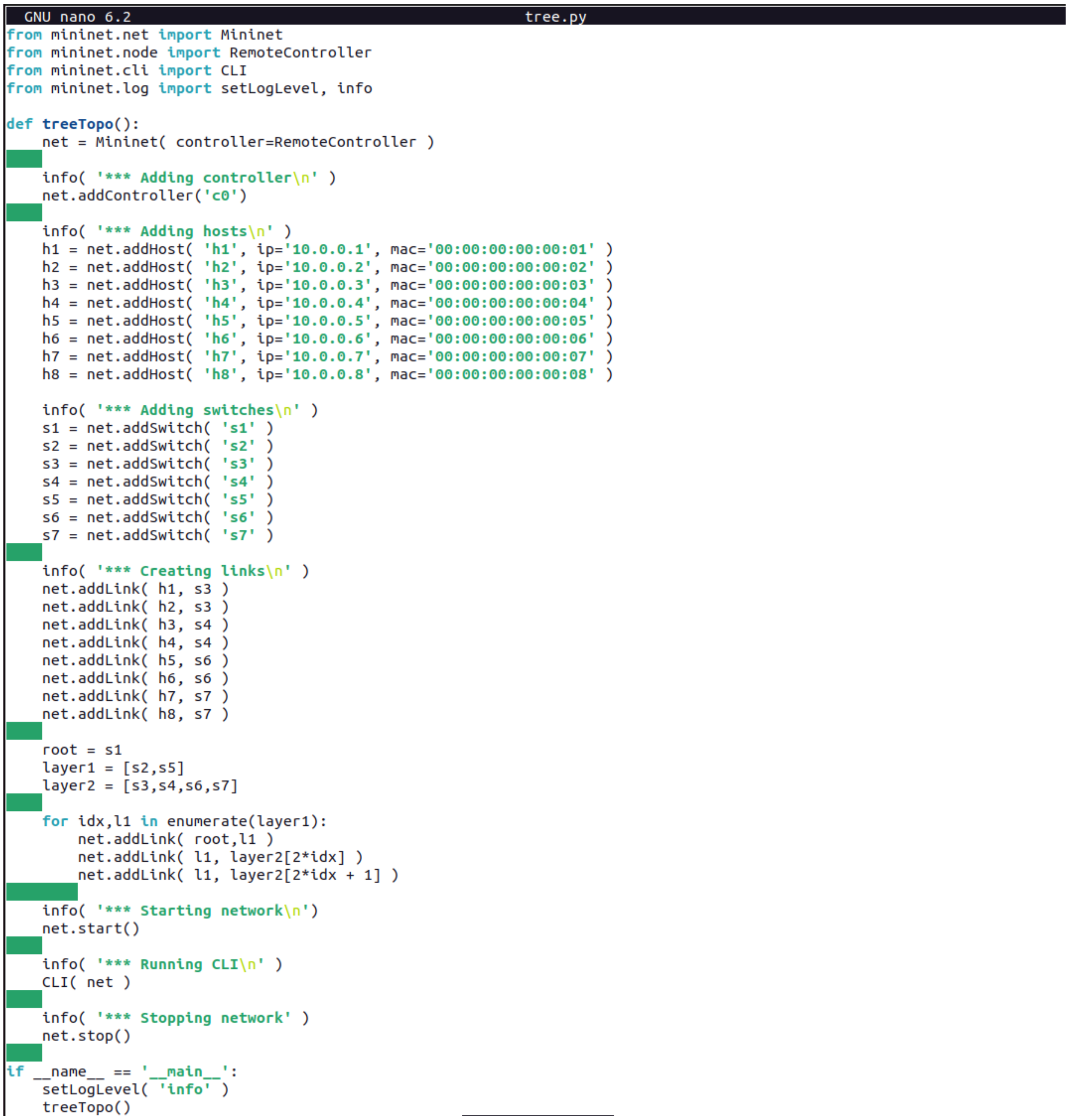
```
ubuntu@ubuntu-virtual-machine:~$ sudo apt install mininet
[sudo] password for ubuntu:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
mininet is already the newest version (2.3.0-1ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 57 not upgraded.
```

Cd into pox/pox/misc and create the following file



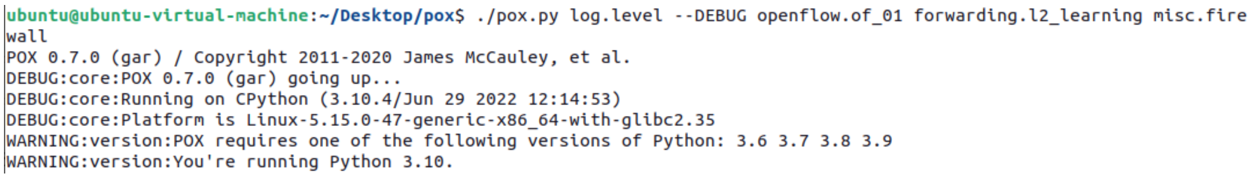
```
  GNU nano 6.2                                        firewall.py
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.addresses import EthAddr

rules = [['00:00:00:00:00:01','00:00:00:00:00:02'],['00:00:00:00:00:02', '00:00:00:00:00:04'],['00:00:00:00:00:08','00>

class SDNFirewall (EventMixin):

    def __init__ (self):
        self.listenTo(core.openflow)

    def _handle_ConnectionUp (self, event):
        for rule in rules:
            block = of.ofp_match()
            block.dl_src = EthAddr(rule[0])
            block.dl_dst = EthAddr(rule[1])
            flow_mod = of.ofp_flow_mod()
            flow_mod.match = block
            event.connection.send(flow_mod)

def launch ():
    core.registerNew(SDNFirewall)
```

Write the following config in any location



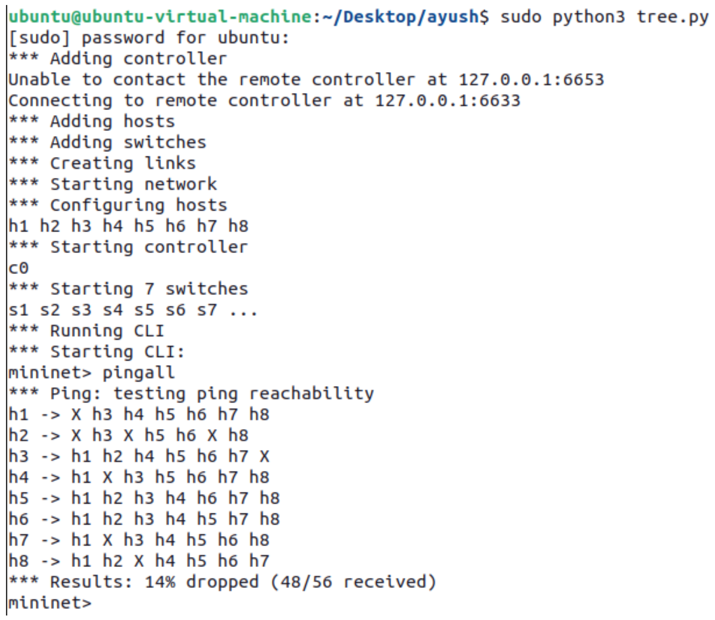
```
  GNU nano 6.2                                        tree.py
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.cli import CLI
from mininet.log import setLogLevel, info

def treeTopo():
    net = Mininet( controller=RemoteController )

    info( '*** Adding controller\n' )
    net.addController('c0')

    info( '*** Adding hosts\n' )
    h1 = net.addHost( 'h1', ip='10.0.0.1', mac='00:00:00:00:00:01' )
    h2 = net.addHost( 'h2', ip='10.0.0.2', mac='00:00:00:00:00:02' )
    h3 = net.addHost( 'h3', ip='10.0.0.3', mac='00:00:00:00:00:03' )
    h4 = net.addHost( 'h4', ip='10.0.0.4', mac='00:00:00:00:00:04' )
    h5 = net.addHost( 'h5', ip='10.0.0.5', mac='00:00:00:00:00:05' )
    h6 = net.addHost( 'h6', ip='10.0.0.6', mac='00:00:00:00:00:06' )
    h7 = net.addHost( 'h7', ip='10.0.0.7', mac='00:00:00:00:00:07' )
    h8 = net.addHost( 'h8', ip='10.0.0.8', mac='00:00:00:00:00:08' )

    info( '*** Adding switches\n' )
    s1 = net.addSwitch( 's1' )
    s2 = net.addSwitch( 's2' )
    s3 = net.addSwitch( 's3' )
    s4 = net.addSwitch( 's4' )
    s5 = net.addSwitch( 's5' )
    s6 = net.addSwitch( 's6' )
    s7 = net.addSwitch( 's7' )

    info( '*** Creating links\n' )
    net.addLink( h1, s3 )
    net.addLink( h2, s3 )
    net.addLink( h3, s4 )
    net.addLink( h4, s4 )
    net.addLink( h5, s6 )
    net.addLink( h6, s6 )
    net.addLink( h7, s7 )
    net.addLink( h8, s7 )

    root = s1
    layer1 = [s2,s5]
    layer2 = [s3,s4,s6,s7]

    for idx,l1 in enumerate(layer1):
        net.addLink( root,l1 )
        net.addLink( l1, layer2[2*idx] )
        net.addLink( l1, layer2[2*idx + 1] )

    info( '*** Starting network\n')
    net.start()

    info( '*** Running CLI\n' )
    CLI( net )

    info( '*** Stopping network' )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    treeTopo()
```

Cd into root pox repo and run the following command



```
ubuntu@ubuntu-virtual-machine:~/Desktop/pox$ ./pox.py log.level --DEBUG openflow.of_01 forwarding.l2_learning misc.fire
wall
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.10.4/Jun 29 2022 12:14:53)
DEBUG:core:Platform is Linux-5.15.0-47-generic-x86_64-with-glibc2.35
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
```

Run the above tree.py

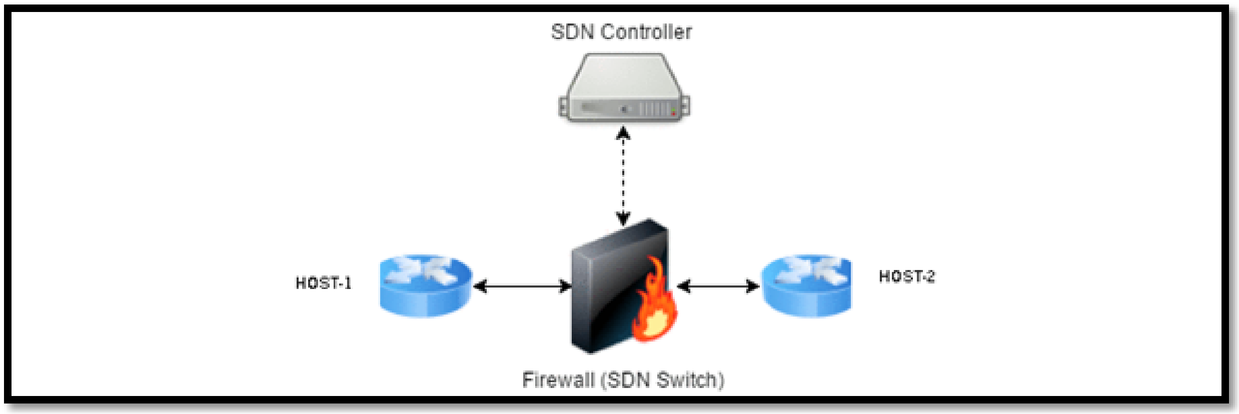


```
ubuntu@ubuntu-virtual-machine:~/Desktop/ayush$ sudo python3 tree.py
[sudo] password for ubuntu:
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts
*** Adding switches
*** Creating links
*** Starting network
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Running CLI
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3 h4 h5 h6 h7 h8
h2 -> X h3 X h5 h6 X h8
h3 -> h1 h2 h4 h5 h6 h7 X
h4 -> h1 X h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 X h3 h4 h5 h6 h8
h8 -> h1 h2 X h4 h5 h6 h7
*** Results: 14% dropped (48/56 received)
mininet>
```

Write Up:

SDN Firewall

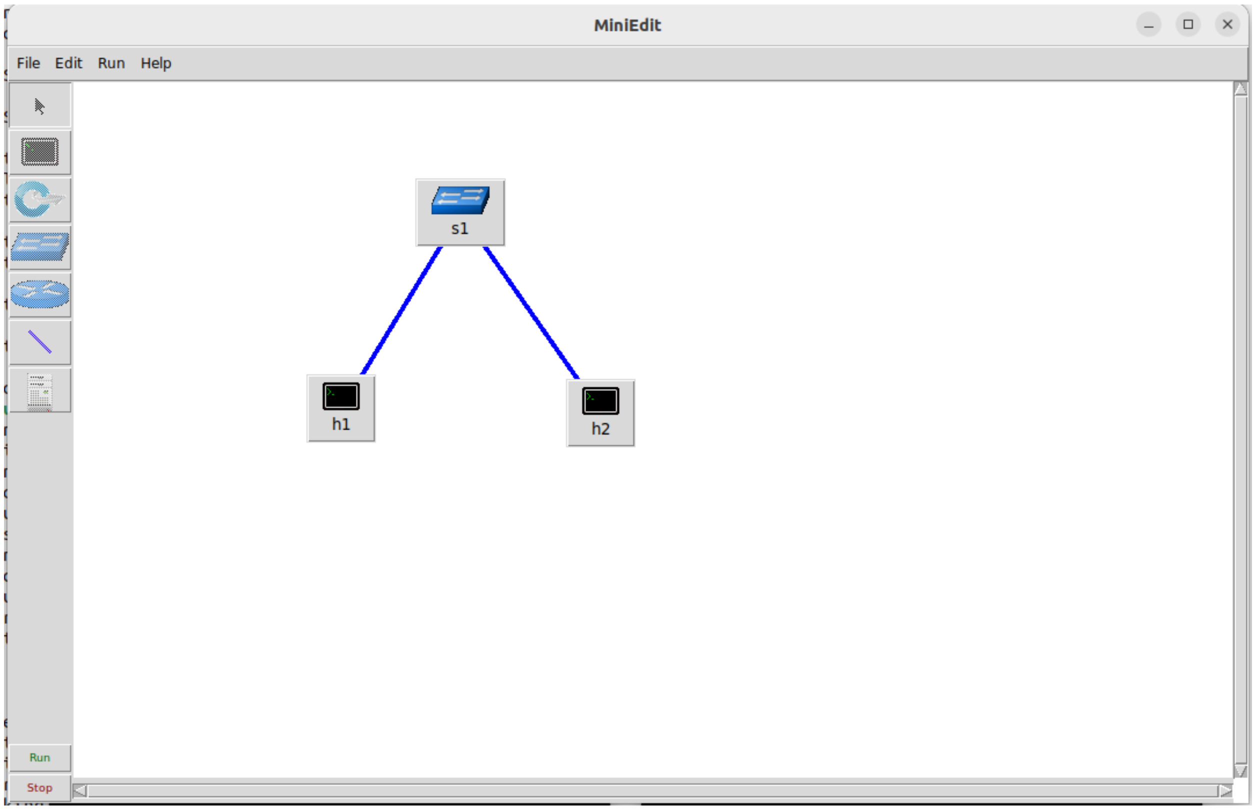
A Firewall can be understood to be something that obstructs traffic coming its way and filters it according to some rules. A general Firewall can be employed to protect a network from the internet. For an SDN based Firewall, we are going to use the OpenFlow controller to filter traffic between hosts according to some rules and accordingly let it pass through or not. The way we will do this is by using the POX controller to establish our required policies or rules (we will come to these later) and filter traffic between hosts using the switches. An example of such an SDN Firewall with two hosts is shown in the figure below:



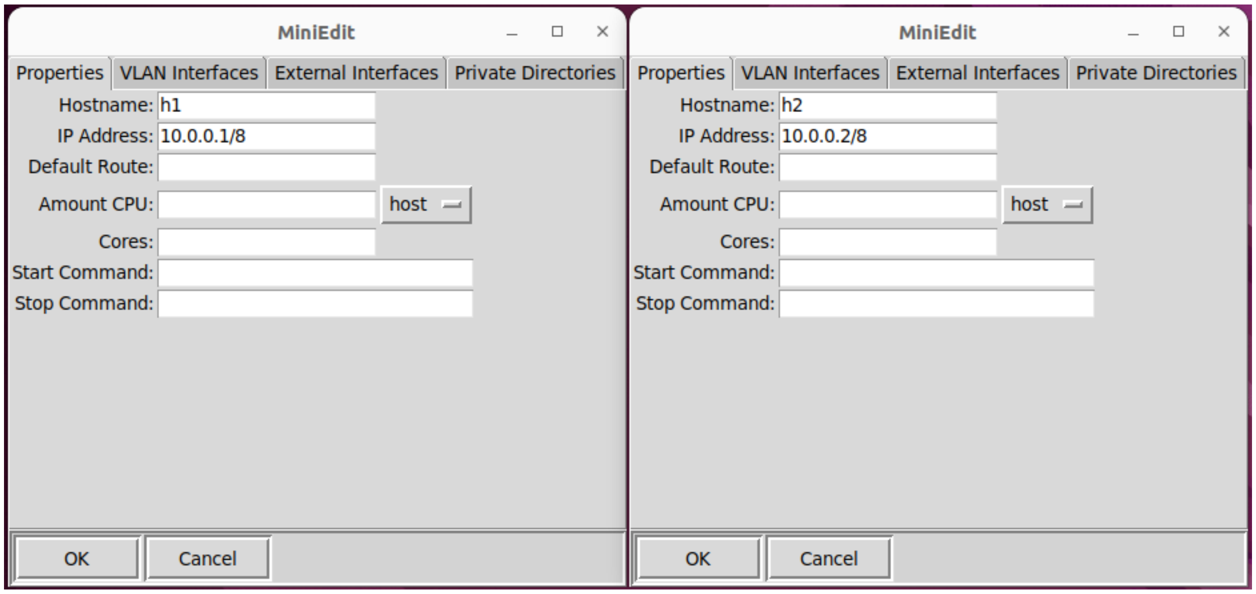
Aim : Implementing Simple Router using mininet.

Steps :

Create the following topology



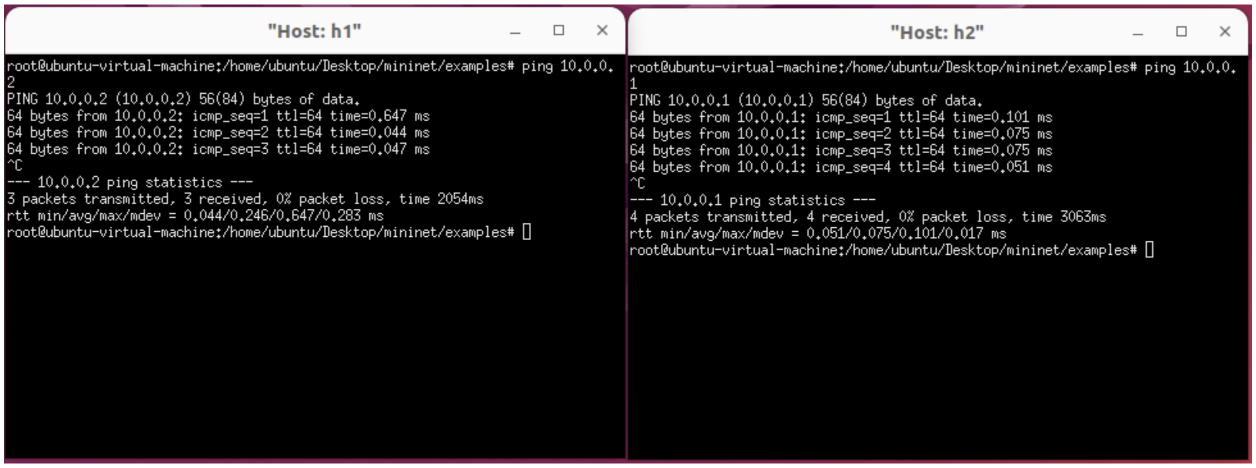
Configure the two hosts



Hostname: h1
IP Address: 10.0.0.1/8
Default Route:
Amount CPU: host
Cores:
Start Command:
Stop Command:

Hostname: h2
IP Address: 10.0.0.2/8
Default Route:
Amount CPU: host
Cores:
Start Command:
Stop Command:

Try pinging the two hosts



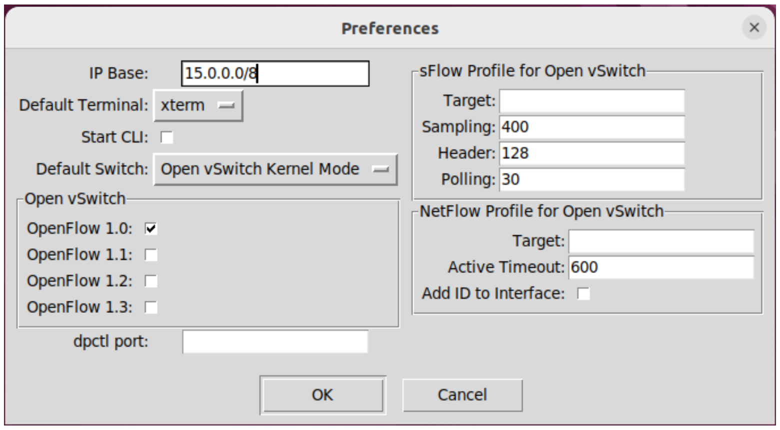
"Host: h1"

```
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# ping 10.0.0.
2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.647 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.047 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2054ms
rtt min/avg/max/mdev = 0.044/0.246/0.647/0.283 ms
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# 
```

"Host: h2"

```
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# ping 10.0.0.
1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.101 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.075 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.051 ms
^C
--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3063ms
rtt min/avg/max/mdev = 0.051/0.075/0.101/0.017 ms
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# 
```

Make the following changes under Edit > Preferences



Preferences

IP Base: 15.0.0.0/8
Default Terminal: xterm
Start CLI: ☐
Default Switch: Open vSwitch Kernel Mode

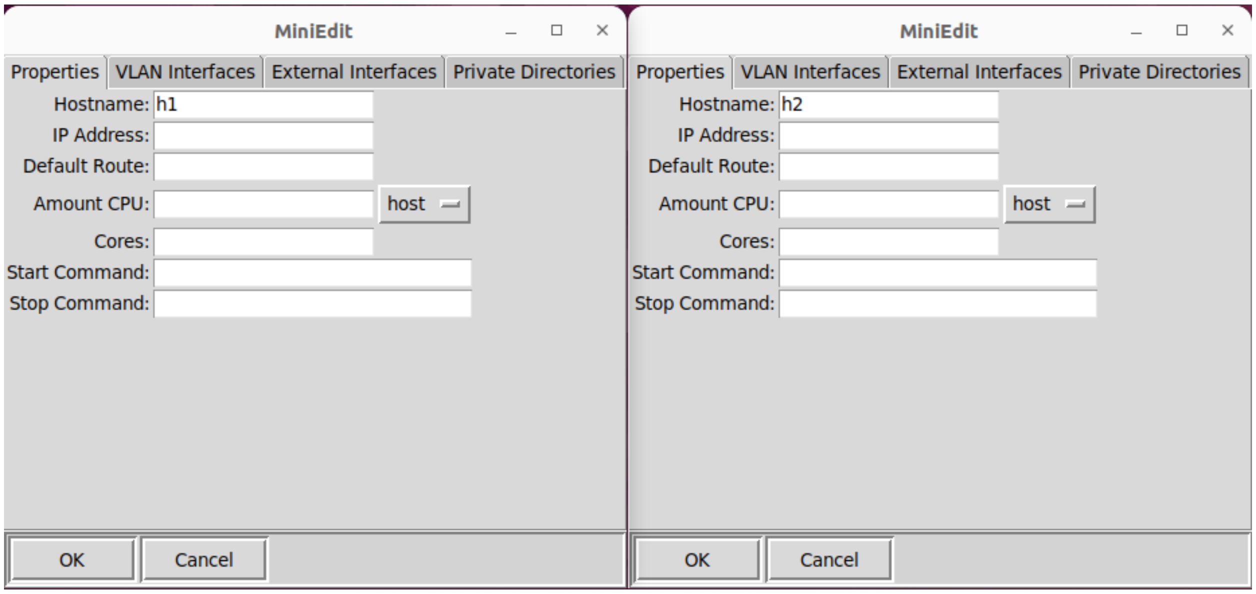
Open vSwitch
OpenFlow 1.0: ✓
OpenFlow 1.1: ☐
OpenFlow 1.2: ☐
OpenFlow 1.3: ☐

dpctl port:

sFlow Profile for Open vSwitch
Target:
Sampling: 400
Header: 128
Polling: 30

NetFlow Profile for Open vSwitch
Target:
Active Timeout: 600
Add ID to Interface: ☐

Make the following changes in the two hosts



MiniEdit

Properties | VLAN Interfaces | External Interfaces | Private Directories

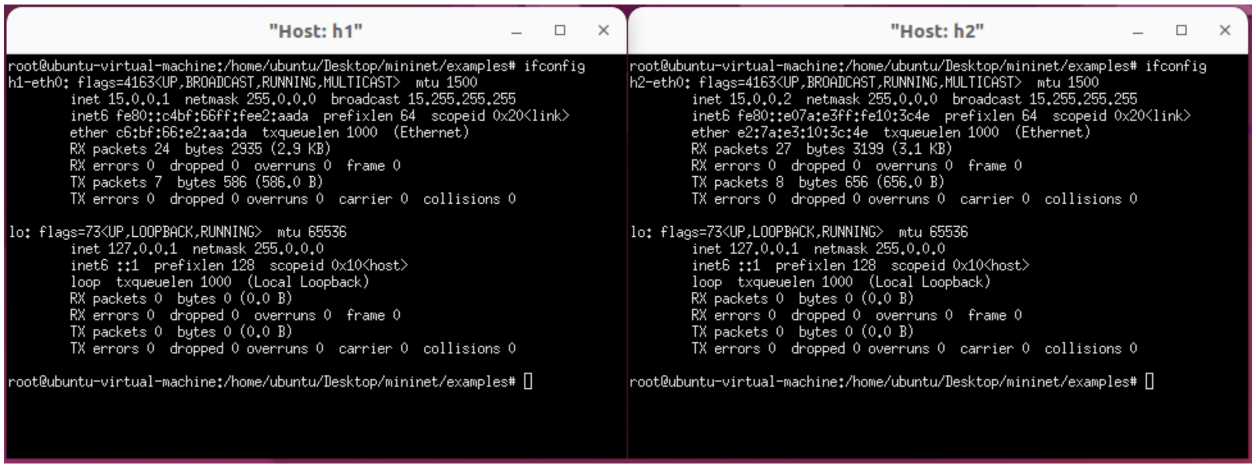
Hostname: h1
IP Address:
Default Route:
Amount CPU: host
Cores:
Start Command:
Stop Command:

MiniEdit

Properties | VLAN Interfaces | External Interfaces | Private Directories

Hostname: h2
IP Address:
Default Route:
Amount CPU: host
Cores:
Start Command:
Stop Command:

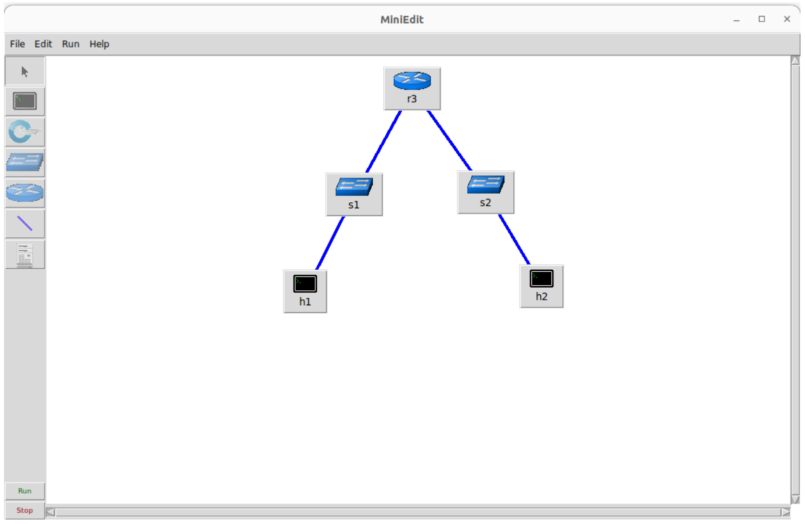
Check network info of the two hosts via ifconfg



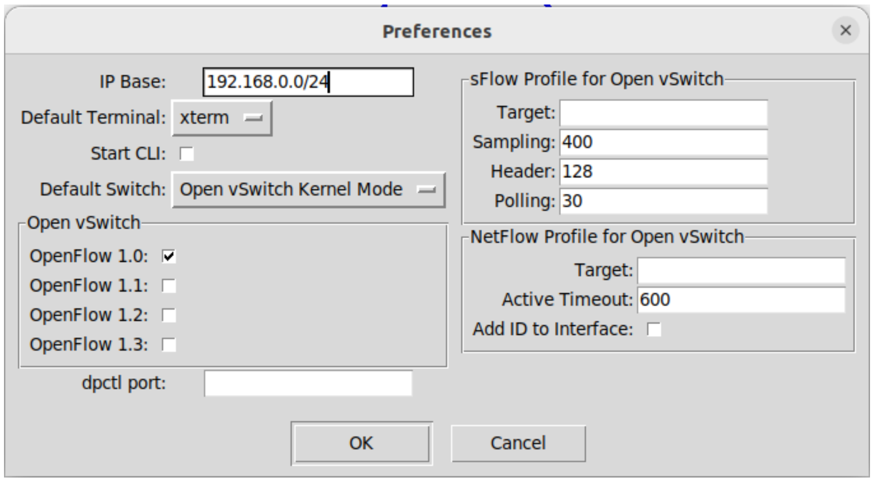
```
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 15.0.0.1  netmask 255.0.0.0  broadcast 15.255.255.255
        inet6 fe80::c4bf:66ff:fee2:aada  prefixlen 64  scopeid 0x20<link>
        ether c6:bf:66:e2:aa:da  txqueuelen 1000  (Ethernet)
        RX packets 24  bytes 2935 (2.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7  bytes 586 (586.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# 
```

```
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 15.0.0.2  netmask 255.0.0.0  broadcast 15.255.255.255
        inet6 fe80::e07a:e3ff:fe10:3c4e  prefixlen 64  scopeid 0x20<link>
        ether e2:7a:e3:10:3c:4e  txqueuelen 1000  (Ethernet)
        RX packets 27  bytes 3199 (3.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 656 (656.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# 
```

Create the following topology



Make the following changes under Edit > Preferences



IP Base: 192.168.0.0/24
Default Terminal: xterm
Start CLI: ☐
Default Switch: Open vSwitch Kernel Mode

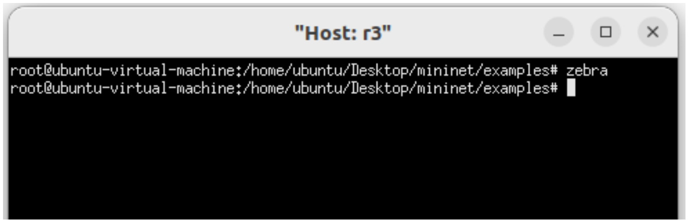
Open vSwitch
OpenFlow 1.0: ☑
OpenFlow 1.1: ☐
OpenFlow 1.2: ☐
OpenFlow 1.3: ☐

dpctl port:

sFlow Profile for Open vSwitch
Target:
Sampling: 400
Header: 128
Polling: 30

NetFlow Profile for Open vSwitch
Target:
Active Timeout: 600
Add ID to Interface: ☐

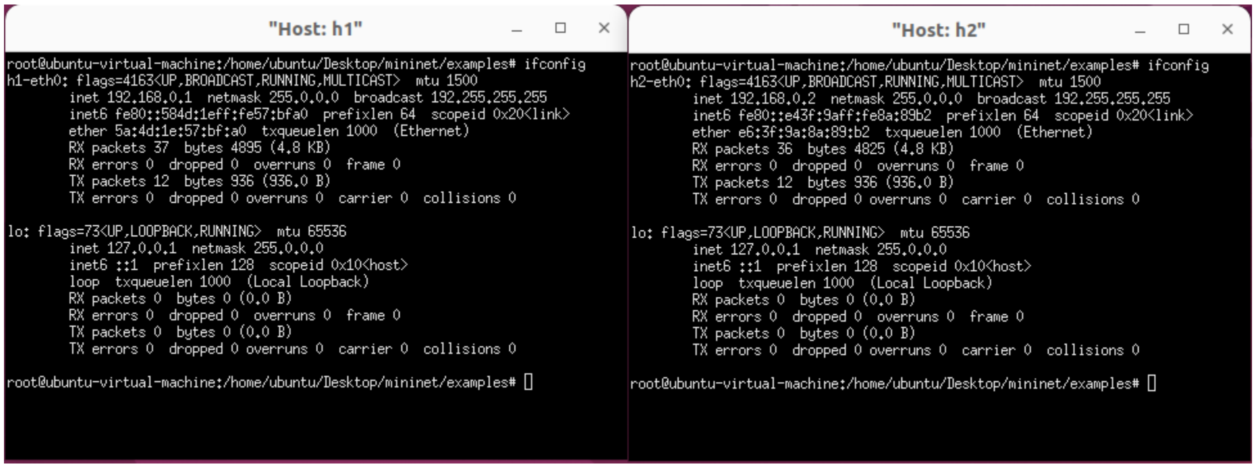
Check whether zebra command is working



"Host: r3"

```
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# zebra
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples#
```

Check network info of the two hosts via ifconfig



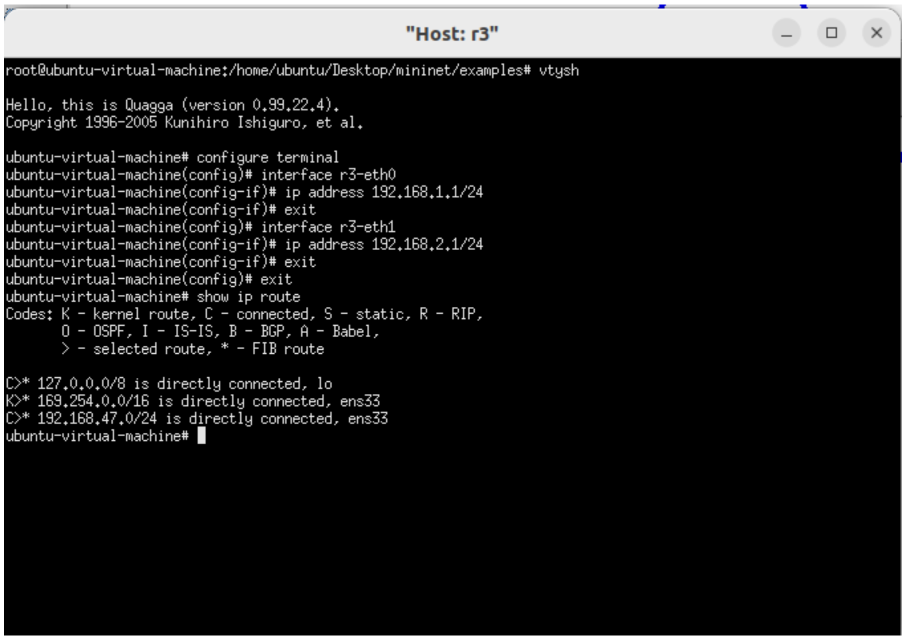
"Host: h1"

```
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.1  netmask 255.0.0.0  broadcast 192.255.255.255
        inet6 fe80::584d:1eff:fe57:bfa0  prefixlen 64  scopeid 0x20<link>
        ether 5a:4d:1e:57:bf:a0  txqueuelen 1000  (Ethernet)
        RX packets 37  bytes 4895 (4.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 936 (936.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples#
```

"Host: h2"

```
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.2  netmask 255.0.0.0  broadcast 192.255.255.255
        inet6 fe80::e43f:9aff:fe8a:89b2  prefixlen 64  scopeid 0x20<link>
        ether e6:3f:9a:8a:89:b2  txqueuelen 1000  (Ethernet)
        RX packets 36  bytes 4825 (4.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 936 (936.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples#
```

Configure the router



"Host: r3"

```
root@ubuntu-virtual-machine:/home/ubuntu/Desktop/mininet/examples# vtysh

Hello, this is Quagga (version 0.99.22.4).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

ubuntu-virtual-machine# configure terminal
ubuntu-virtual-machine(config)# interface r3-eth0
ubuntu-virtual-machine(config-if)# ip address 192.168.1.1/24
ubuntu-virtual-machine(config-if)# exit
ubuntu-virtual-machine(config)# interface r3-eth1
ubuntu-virtual-machine(config-if)# ip address 192.168.2.1/24
ubuntu-virtual-machine(config-if)# exit
ubuntu-virtual-machine(config)# exit
ubuntu-virtual-machine# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
K>* 169.254.0.0/16 is directly connected, ens33
C>* 192.168.47.0/24 is directly connected, ens33
ubuntu-virtual-machine#
```

Write Up:

Sometimes simulations are not possible or not practical, and network experiments must be run on actual machines. One can always use a set of interconnected virtual machines, but even pared-down virtual machines consume sufficient resources that it is hard to create a network of more than a handful of nodes. Mininet is a system that supports the creation of lightweight logical nodes that can be connected into networks. These nodes are sometimes called containers, or, more accurately, network namespaces. Virtual-machine technology is not used. These containers consume sufficiently few resources that networks of over a thousand nodes have been created, running on a single laptop. While Mininet was originally developed as a testbed for software-defined networking (3.4 Software-Defined Networking), it works just as well for demonstrations and experiments involving traditional networking.

A Mininet container is a process (or group of processes) that no longer has access to all the host system's "native" network interfaces, much as a process that has executed the chroot() system call no longer has access to the full filesystem. Mininet containers then are assigned virtual Ethernet interfaces (see the ip-link man page entries for veth), which are connected to other containers through virtual Ethernet links. The use of veth links ensures that the virtual links behave like Ethernet, though it may be necessary to disable TSO (17.5 TCP Offloading) to view Ethernet packets in WireShark as they would appear on the (virtual) wire. Any process started within a Mininet container inherits the container's view of network interfaces.

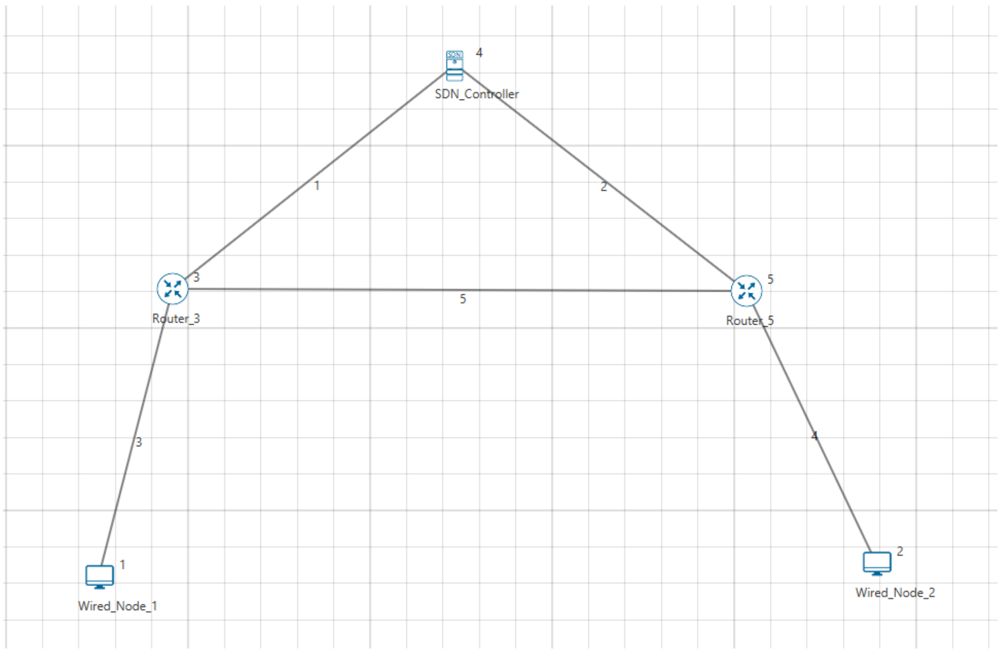
For efficiency, Mininet containers all share the same filesystem by default. This makes setup simple, but sometimes causes problems with applications that expect individualized configuration files in specified locations. Mininet containers can be configured so that each container has at least one private directory, eg for configuration files. See 30.6 Quagga Routing and BGP for an example, though mostly we avoid this feature.

Mininet is a form of network emulation, as opposed to simulation. An important advantage of emulation is that all network software, at any layer, is simply run "as is". In a simulator environment, on the other hand, applications and protocol implementations need to be ported to run within the simulator before they can be used. A drawback of emulation is that as the network gets large and complex the emulation may slow down. In particular, it is not possible to emulate link speeds faster than the underlying hardware can support. (It is also not possible to emulate non-Linux network software.)

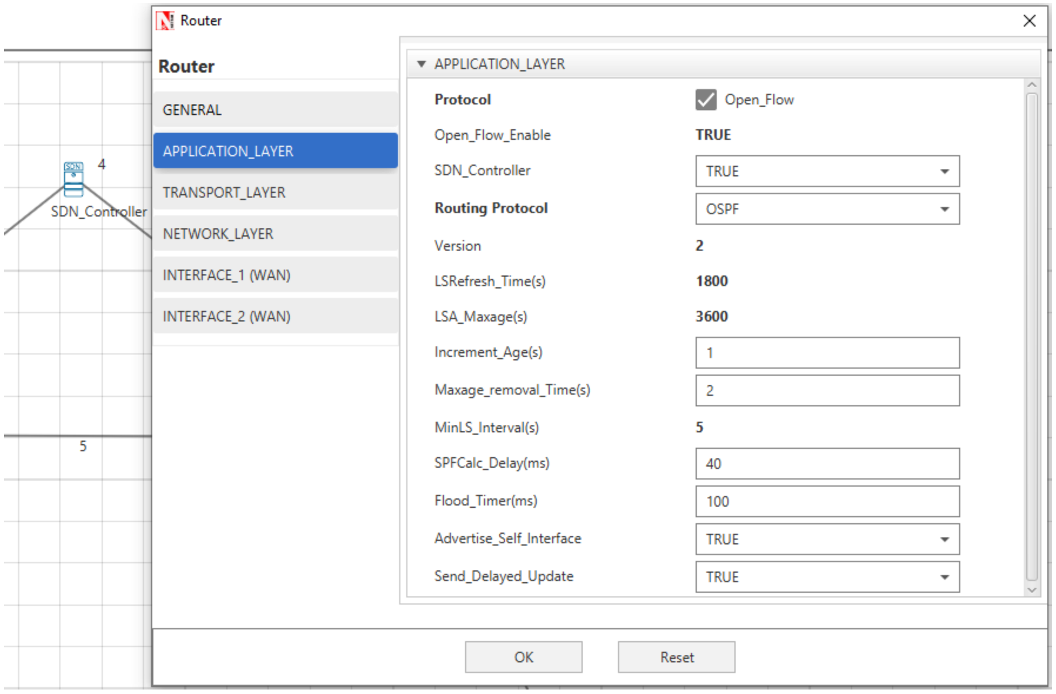
Practical No 8

Aim : NetSim

Create the following config

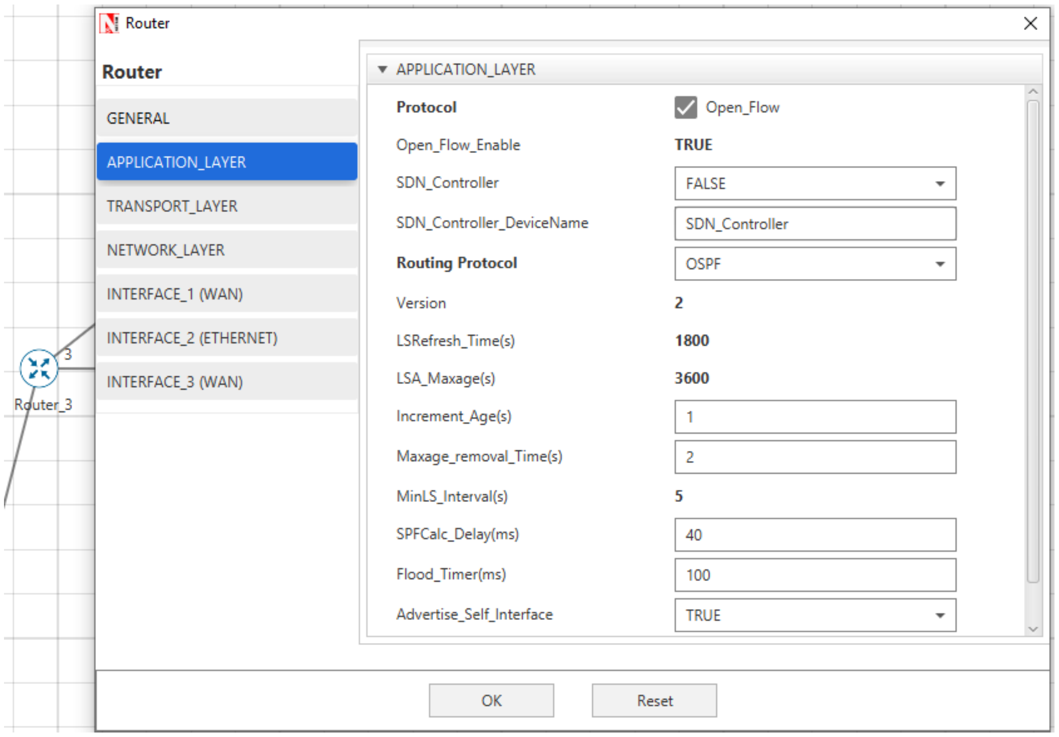


SDN Controller > Properties > Application Layer [TRUE, OpenFlow checked]



Protocol	✓ Open_Flow
Open_Flow_Enable	TRUE
SDN_Controller	TRUE
Routing Protocol	OSPF
Version	2
LSRefresh_Time(s)	1800
LSA_Maxage(s)	3600
Increment_Age(s)	1
Maxage_removal_Time(s)	2
MinLS_Interval(s)	5
SPFCalc_Delay(ms)	40
Flood_Timer(ms)	100
Advertise_Self_Interface	TRUE
Send_Delayed_Update	TRUE

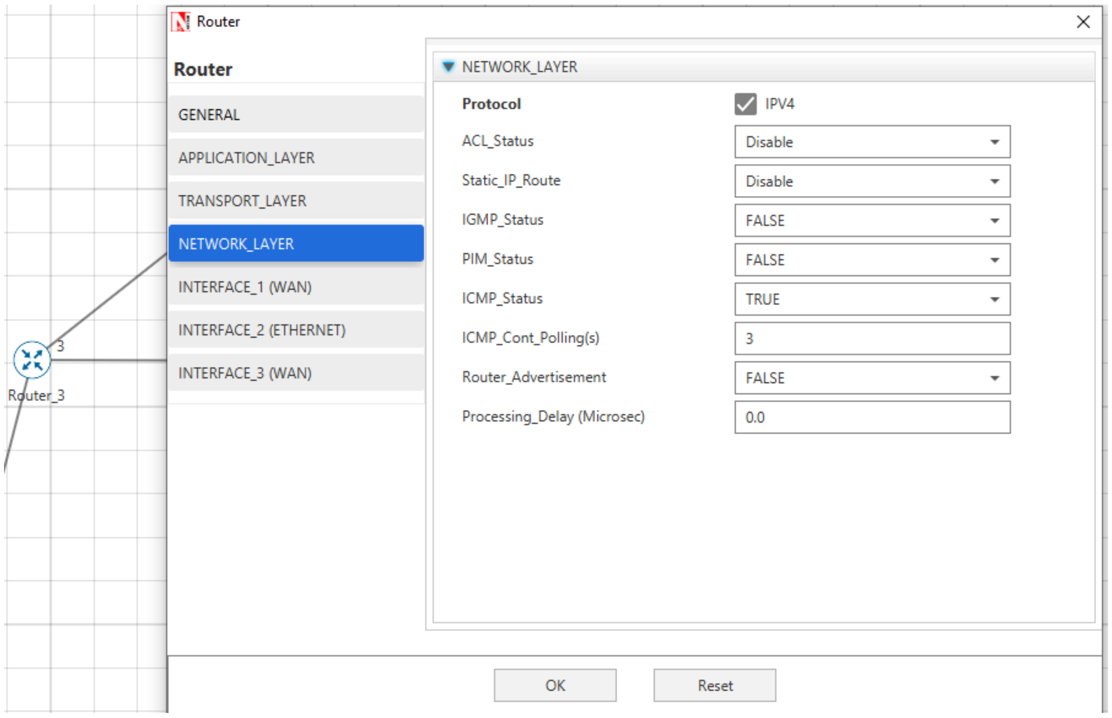
Router > App Layer > [Open flow check, False]



APPLICATION_LAYER

Protocol	✓ Open_Flow
Open_Flow_Enable	**TRUE**
SDN_Controller	FALSE
SDN_Controller_DeviceName	SDN_Controller
Routing Protocol	OSPF
Version	**2**
LSRefresh_Time(s)	**1800**
LSA_Maxage(s)	**3600**
Increment_Age(s)	1
Maxage_removal_Time(s)	2
MinLS_Interval(s)	**5**
SPFCalc_Delay(ms)	40
Flood_Timer(ms)	100
Advertise_Self_Interface	TRUE

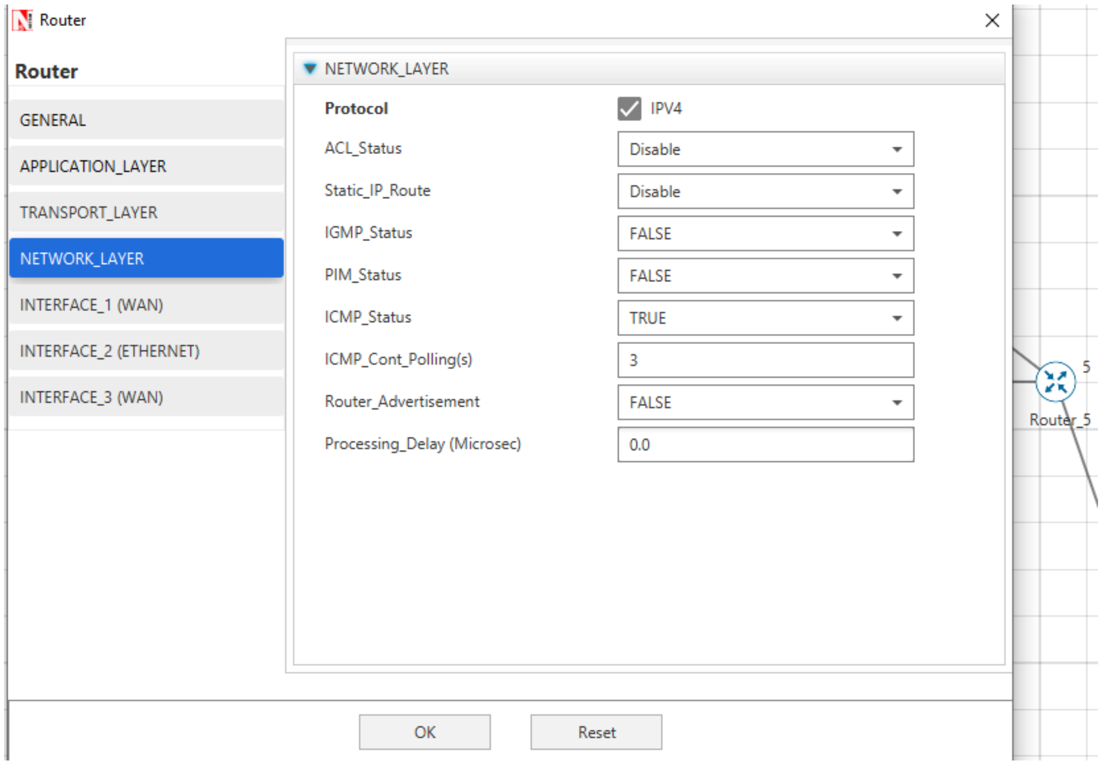
Router > Network Layer > [ICMP true]



NETWORK_LAYER

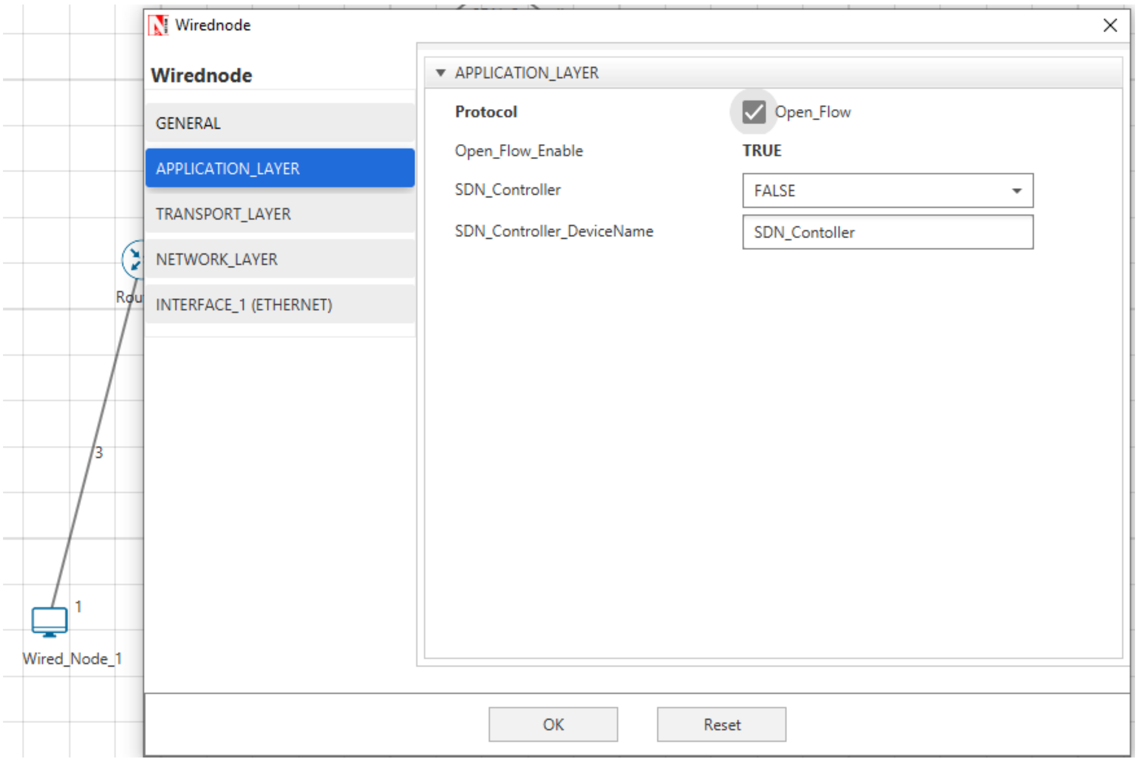
Protocol	✓ IPV4
ACL_Status	Disable
Static_IP_Route	Disable
IGMP_Status	FALSE
PIM_Status	FALSE
ICMP_Status	TRUE
ICMP_Cont_Polling(s)	3
Router_Advertisement	FALSE
Processing_Delay (Microsec)	0.0

Same for the router 5



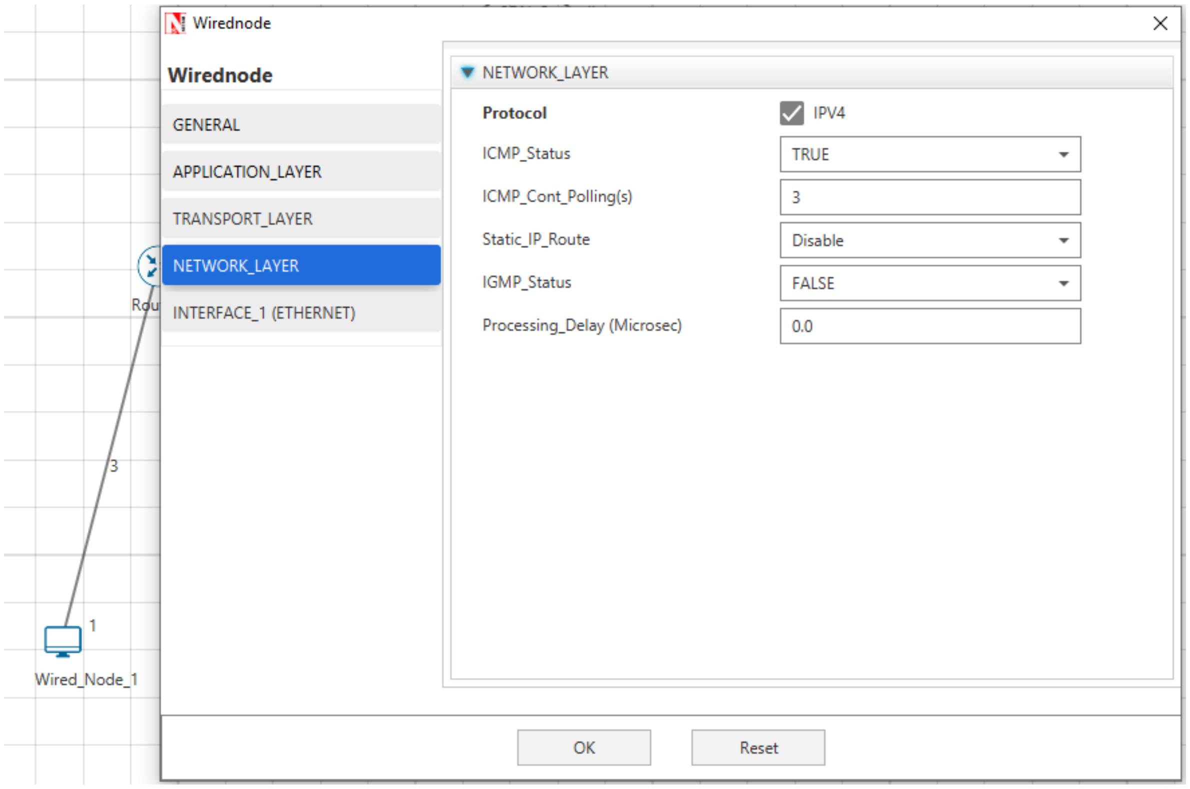
Protocol	✓ IPV4
ACL_Status	Disable
Static_IP_Route	Disable
IGMP_Status	FALSE
PIM_Status	FALSE
ICMP_Status	TRUE
ICMP_Cont_Polling(s)	3
Router_Advertisement	FALSE
Processing_Delay (Microsec)	0.0

Wired Node > App layer > [Open FLow Check, SDN controller false and name it]



Protocol	✓ Open_Flow
Open_Flow_Enable	TRUE
SDN_Controller	FALSE
SDN_Controller_DeviceName	SDN_Contoller

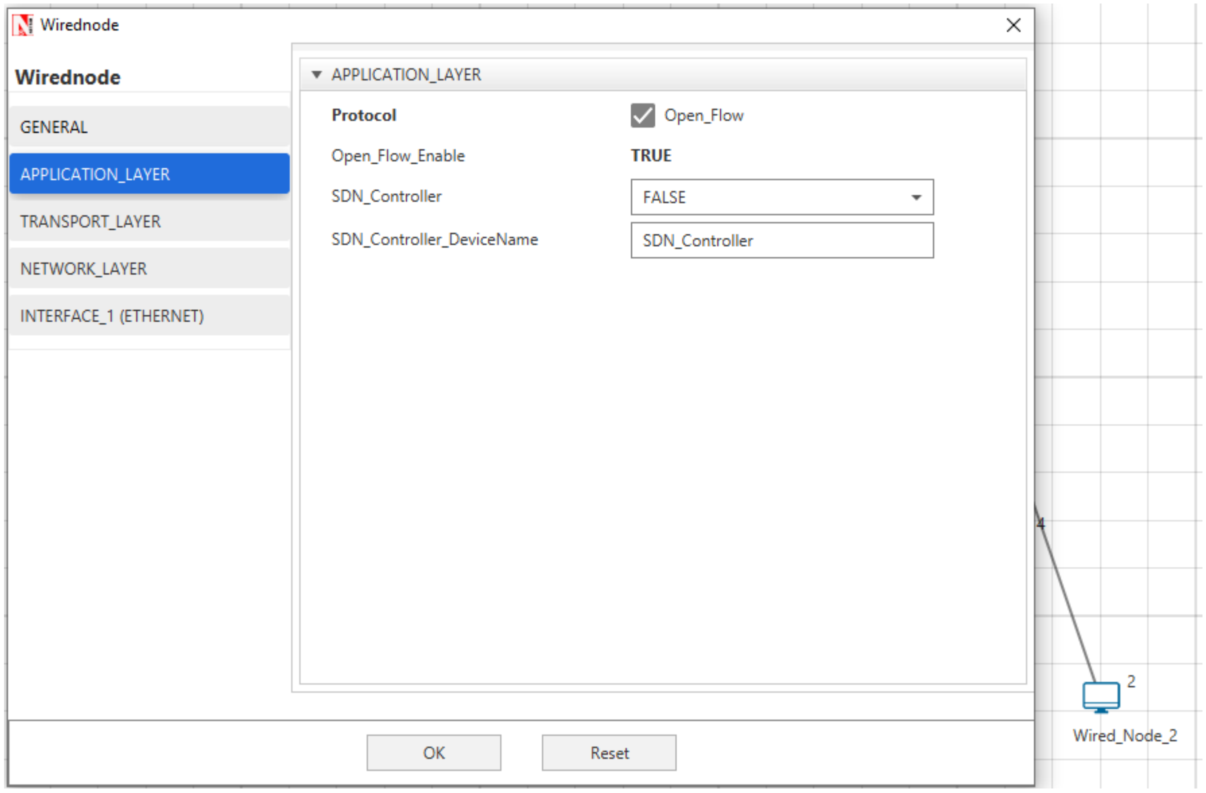
Wired Node > ICMP status > [True]



NETWORK_LAYER

Protocol	✓ IPV4
ICMP_Status	TRUE
ICMP_Cont_Polling(s)	3
Static_IP_Route	Disable
IGMP_Status	FALSE
Processing_Delay (Microsec)	0.0

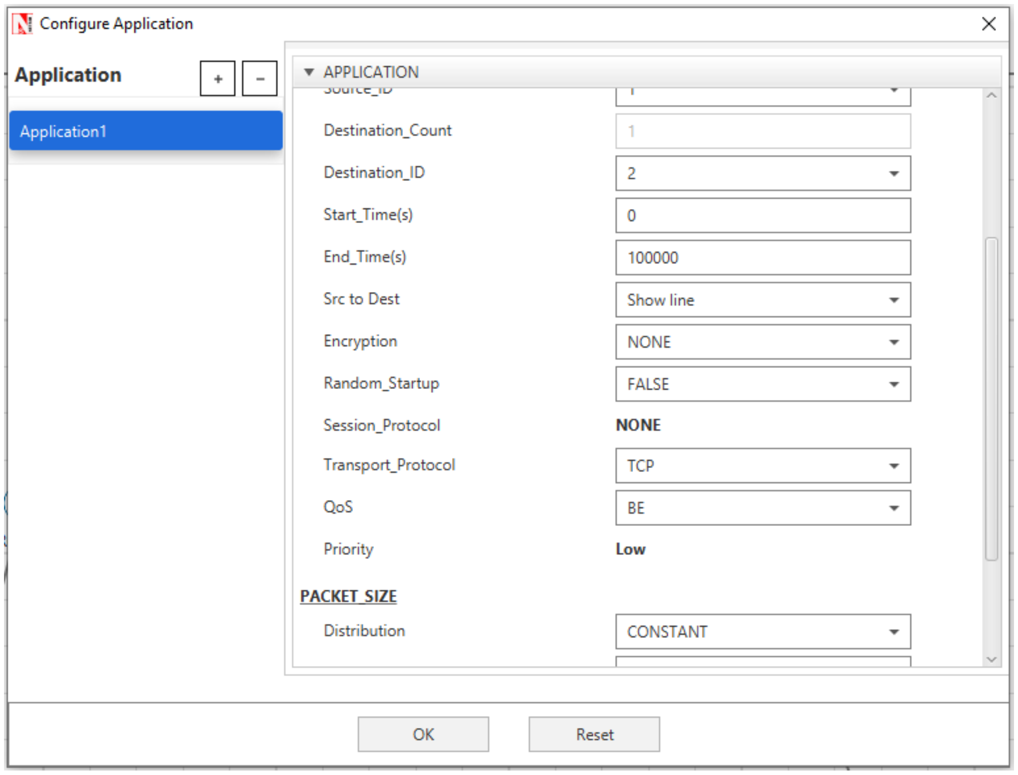
Same for wired node 2



APPLICATION_LAYER

Protocol	✓ Open_Flow
Open_Flow_Enable	**TRUE**
SDN_Controller	FALSE
SDN_Controller_DeviceName	SDN_Controller

Application > Transport protocol > TCP


Configure Application

Application

Application1

APPLICATION

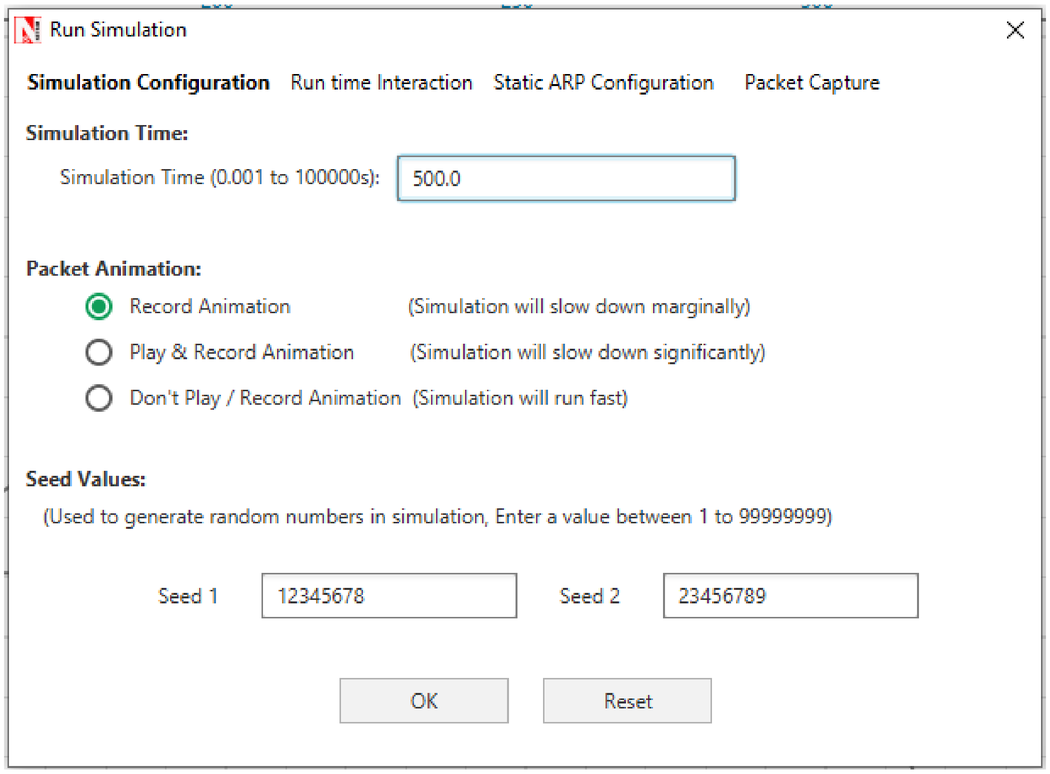
Destination_Count	1
Destination_ID	2
Start_Time(s)	0
End_Time(s)	100000
Src to Dest	Show line
Encryption	NONE
Random_Startup	FALSE
Session_Protocol	NONE
Transport_Protocol	TCP
QoS	BE
Priority	Low

PACKET SIZE

Distribution	CONSTANT

OK Reset

Run > Sim Config > Sim time > 500


Run Simulation

Simulation Configuration | Run time Interaction | Static ARP Configuration | Packet Capture

Simulation Time:

Simulation Time (0.001 to 100000s): 500.0

Packet Animation:

[✓ Record Animation] (Simulation will slow down marginally)

Play & Record Animation (Simulation will slow down significantly)

Don't Play / Record Animation (Simulation will run fast)

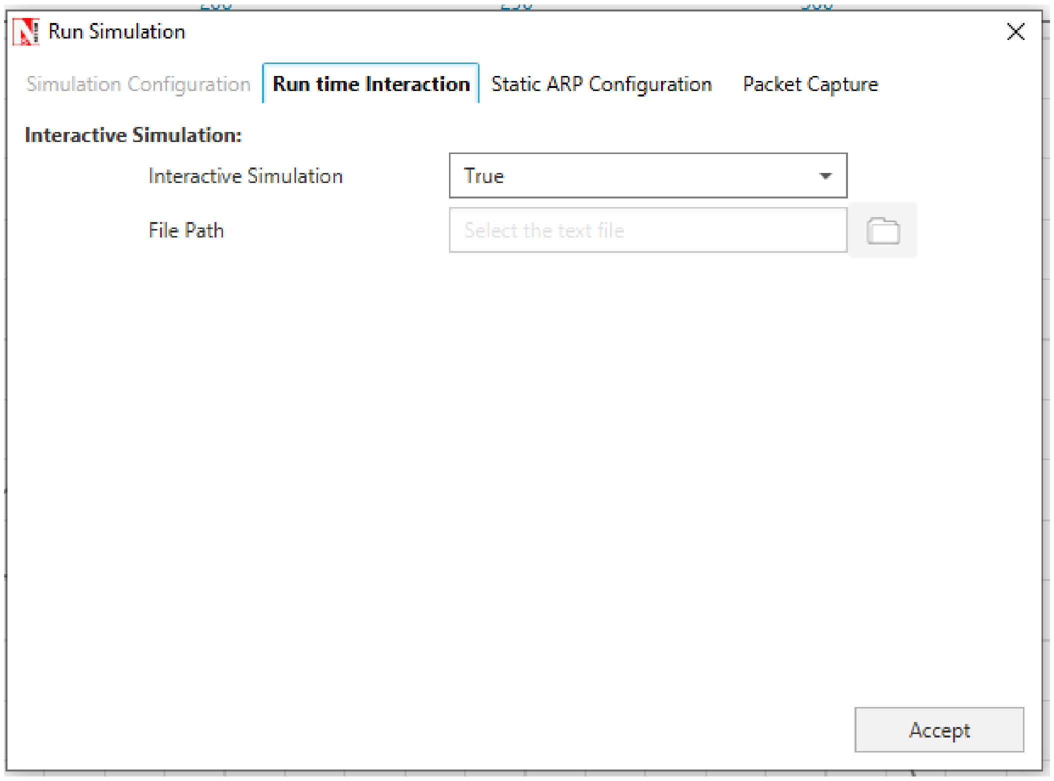
Seed Values:

(Used to generate random numbers in simulation, Enter a value between 1 to 99999999)

Seed 1: 12345678 Seed 2: 23456789

OK Reset

Run > Sim Config > Run time Int > 500


Run Simulation

Simulation Configuration | Run time Interaction | Static ARP Configuration | Packet Capture

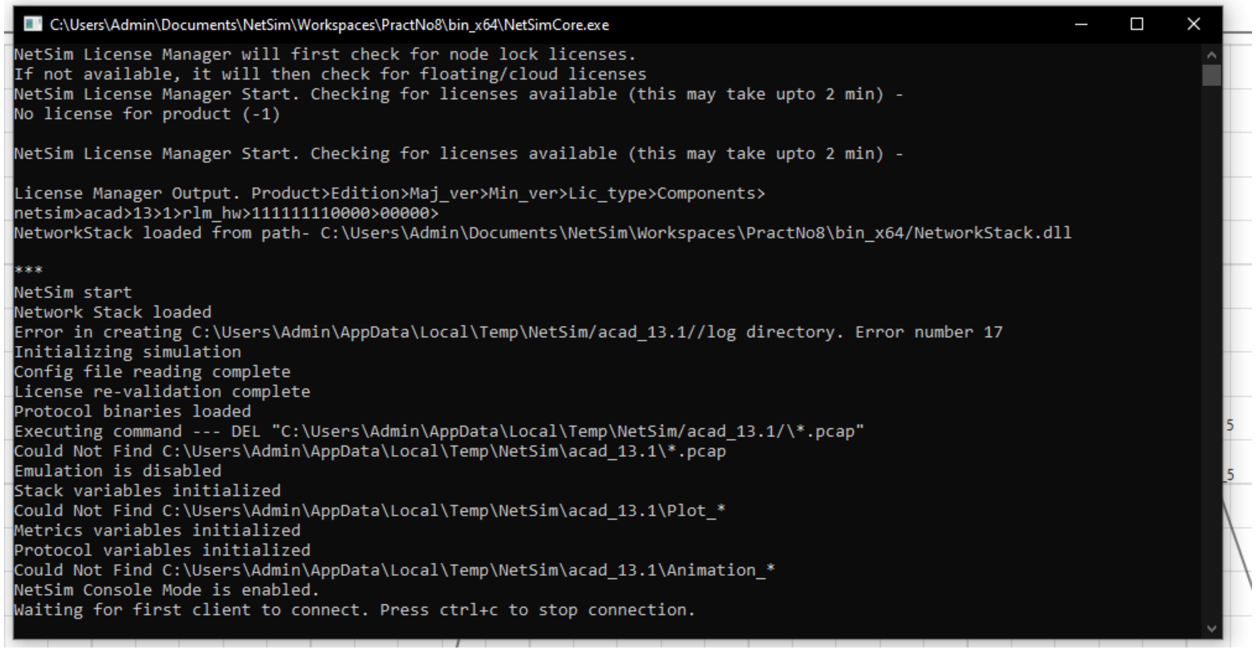
Interactive Simulation:

Interactive Simulation: True

File Path: Select the text file

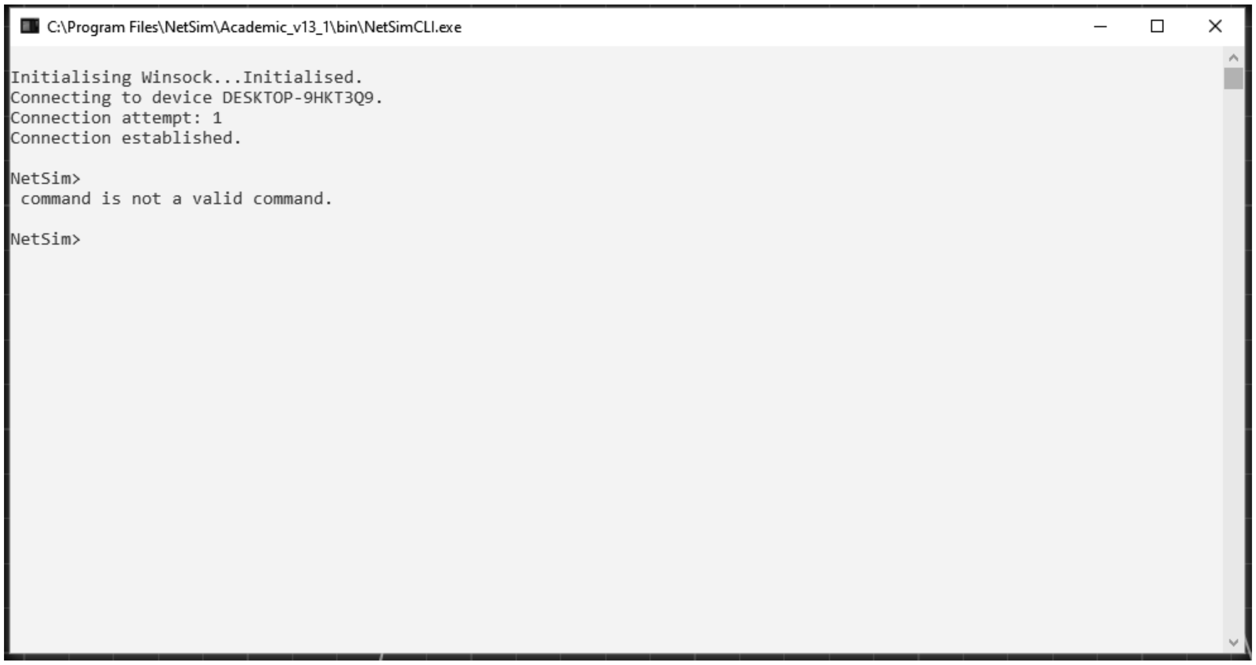
Accept

Don’t Close this

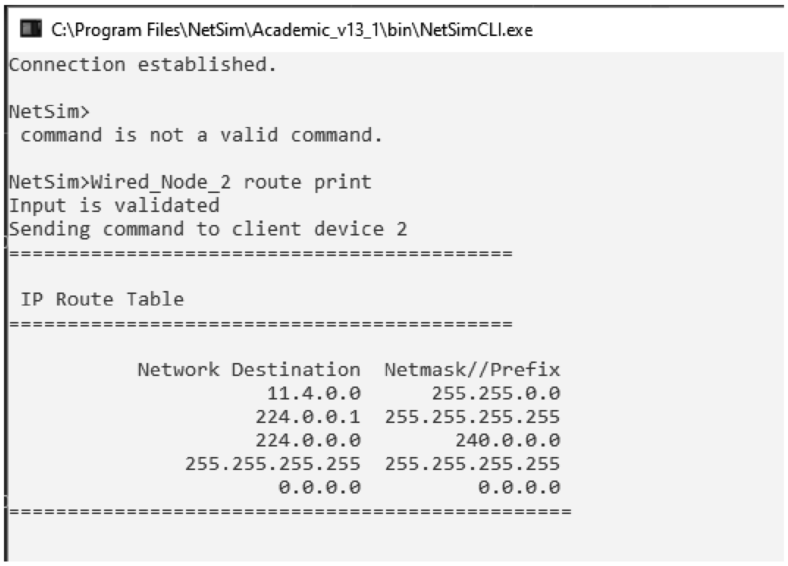

C:\Users\Admin\Documents\NetSim\Workspaces\PractNo8\bin_x64\NetSimCore.exe

```
NetSim License Manager will first check for node lock licenses.
If not available, it will then check for floating/cloud licenses
NetSim License Manager Start. Checking for licenses available (this may take upto 2 min) -
No license for product (-1)

NetSim License Manager Start. Checking for licenses available (this may take upto 2 min) -

License Manager Output. Product>Edition>Maj_ver>Min_ver>Lic_type>Components>
netsim>acad>13>1>rlm_hw>111111110000>00000>
NetworkStack loaded from path- C:\Users\Admin\Documents\NetSim\Workspaces\PractNo8\bin_x64/NetworkStack.dll

***
NetSim start
Network Stack loaded
Error in creating C:\Users\Admin\AppData\Local\Temp\NetSim/acad_13.1//log directory. Error number 17
Initializing simulation
Config file reading complete
License re-validation complete
Protocol binaries loaded
Executing command --- DEL "C:\Users\Admin\AppData\Local\Temp\NetSim/acad_13.1/\*.pcap"
Could Not Find C:\Users\Admin\AppData\Local\Temp\NetSim\acad_13.1\*.pcap
Emulation is disabled
Stack variables initialized
Could Not Find C:\Users\Admin\AppData\Local\Temp\NetSim\acad_13.1\Plot_*
Metrics variables initialized
Protocol variables initialized
Could Not Find C:\Users\Admin\AppData\Local\Temp\NetSim\acad_13.1\Animation_*
NetSim Console Mode is enabled.
Waiting for first client to connect. Press ctrl+c to stop connection.
```

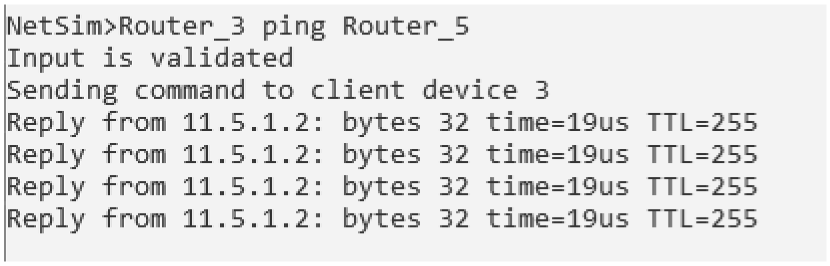
SDN controller > Net sim console



```
Initialising Winsock...Initialised.
Connecting to device DESKTOP-9HKT3Q9.
Connection attempt: 1
Connection established.

NetSim>
 command is not a valid command.

NetSim>
```



```
Connection established.

NetSim>
 command is not a valid command.

NetSim>Wired_Node_2 route print
Input is validated
Sending command to client device 2
===========================================

 IP Route Table
===========================================

           Network Destination  Netmask//Prefix
                      11.4.0.0      255.255.0.0
                     224.0.0.1  255.255.255.255
                     224.0.0.0        240.0.0.0
               255.255.255.255  255.255.255.255
                       0.0.0.0          0.0.0.0
================================================
```



```
NetSim>Router_3 ping Router_5
Input is validated
Sending command to client device 3
Reply from 11.5.1.2: bytes 32 time=19us TTL=255
Reply from 11.5.1.2: bytes 32 time=19us TTL=255
Reply from 11.5.1.2: bytes 32 time=19us TTL=255
Reply from 11.5.1.2: bytes 32 time=19us TTL=255
```